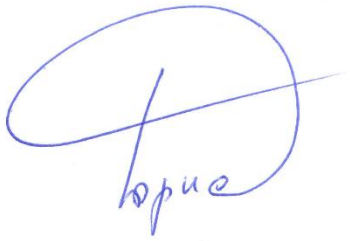
Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

На правах рукописи



Фарков Михаил Александрович

Разработка алгоритмов выполнения молекулярного докинга с использованием графических процессоров

05.13.17 — Теоретические основы информатики

ДИССЕРТАЦИЯ
на соискание учёной степени
кандидата технических наук

Научный руководитель:
доктор технических наук,
профессор А.И. Легалов

Красноярск 2016

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Актуальность исследования.Широкий спектр существующих прикладных задач требует применения сложных вычислительных алгоритмов и методов, предъявляющих высокие требования к вычислительным ресурсам. Для их эффективной реализации используются высокопроизводительные вычислительные системы, в том числе гетерогенные, включающие в себя специализированные процессоры, например, графические. Перенос существующих решений для центрального процессора на гетерогенные вычислительные системы без изменений невозможен в силу значительных различий в программно-аппаратной архитектуре. Поэтому актуальной является задача подбора подходящих алгоритмов и их преобразования с учётом особенностей информационных процессов, протекающийв высокопроизводительных вычислительных системах, и специфики прикладной задачи.

Одной из таких прикладных задач, имеющих важное народно-хозяйственное значение, является выполнение молекулярного докинга, основная область применения которого – разработка лекарственных препаратов. Разработка и тестирование одного лекарственного препарата занимает несколько лет и требует вложения в размере десятков миллионов долларов [1,2]. Весь процесс разработки можно разделить на три основных этапа: подбор перспективных соединений, т.е. испытания *invitro*; доклинические испытания соединений на животных с целью оценки возможных побочных эффектов и общей эффективности; завершающий этап – клинические испытания на человеке, т.е. испытания *invivo*. Первый этап может занимать, по различным оценкам, до 3 лет. Этап является ресурсоёмким, так как требуется отобрать небольшое количество перспективных химических соединений (кандидатов в лекарства) из набора, состоящего из нескольких миллионов веществ, что требует существенных финансовых затрат на реагенты для проведения химических реакций, а также

существенных временных затрат на проведение каждого теста. Именно на первом этапе перспективным является использование компьютерных моделей (*insilico*), что позволяет значительным образом снизить количество химических реакций *invitro* с нескольких миллионов до нескольких десятков для наиболее перспективных соединений, вследствие чего происходит снижение как финансовых, так и временных затрат [5].

Исследования в области молекулярного докинга ведутся уже более 30 лет, и разработано значительное количество разнообразных методов выполнения моделирования, учитывающих различные комбинации аспектов молекулярного взаимодействия и ориентированных на различные классы соединений [25, 26, 28, 29, 30, 35, 36, 39-41, 47-51]. Внедрение перспективных параллельных архитектур, таких как ПЛИС, MIC и графических процессоров, позволит эффективнее и быстрее выполнять молекулярный докинг [90,91,94-112]. Следует отметить, что разработка методов, алгоритмов и подходов для выполнения молекулярного-докинга существенно отстаёт от развития современных перспективных параллельных архитектур. Так, по состоянию на 2016 год лишь несколько программ для молекулярного докинга используют возможности многоядерных центральных процессоров; единицы используют возможности кластеров и массивно параллельных систем (*MPP*), а те, что используют, зачастую реализованы на устаревших библиотеках [3]. Возможности гетерогенных вычислительных систем применяются ещё реже. Вопросы реализации молекулярного докинга для программируемых логических интегральных схем (ПЛИС, *FPGA*) представленыв работах [2, 4, 5]; для графических процессоров – в работах [2, 5-10]. При этом рассмотренные версии программ для графических процессоров реализуют самые простые методы молекулярного докинга, учитывающие только структурную комплементарность молекулярных соединений, что обусловлено их ориентированностью на белок-белковое взаимодействие. Существует всего лишь несколько реализаций для графических процессоров, ориентированных на лиганд-белковое взаимодействие, которые либо не принесли существенного ускорения в процесс выполнения докинга, либо

недостаточно полно используют ресурсы графических процессоров [7-10]. В этом свете наличие значительных параллельных вычислительных ресурсов графических процессоров делает их привлекательной платформой для проведения исследований в этой области, проработки теоретической базы для выполнения моделирования и апробации разработанных алгоритмов.

В основе подавляющего большинства методов лиганд-белкового докинга лежат методы численной оптимизации. Разработка новых методов, алгоритмов и программного обеспечения для численных методов оптимизации, ориентированных на использование ресурсов графических процессоров, позволит повысить эффективность выполнения лиганд-белкового докинга, а также решения широкого круга других прикладных задач фармакологии, биоинформатики, математической физики, экономики и ядерной физики, требующих интенсивного применения численных методов оптимизации в процессе вычислений.

Также следует отметить, что интерес и актуальность имеют исследования, ориентированные на адаптацию существующих численных методов оптимизации под современные высокопроизводительные платформы, среди которых отдельно следует выделить гетерогенные вычислительные системы, использующие графические процессоры. Реализация методов под подобные платформы позволит существенным образом повысить эффективность в соотношении к вычислительным, временным и финансовым затратам за счёт высокой производительности графических процессоров, сосредоточенной в компактном форм-факторе.

Целью работы является разработка алгоритмов реализации численных методов оптимизации на базе гетерогенных вычислительных систем, использующих графические процессоры, и решение с их помощью задачи молекулярного лиганд-белкового докинга.

Для достижения поставленной цели необходимо решить следующие **задачи**:

1. Провести анализ существующих методов и алгоритмов решения задачи молекулярного докинга и возможностей эффективной реализации этих методов на современных высокопроизводительных гетерогенных системах.

2. На основе проведенного анализа предложить модели и алгоритмы, адаптированные для эффективной реализации выбранных методов на графических процессорах.

3. Используя предложенные модели и алгоритмы, разработать программы, обеспечивающие решение поставленной задачи молекулярного лиганд-белкового докинга.

4. Провести анализ эффективности предложенных алгоритмов на примерах рассматриваемой предметной области.

Объектом исследования являются информационные процессы, протекающие в гетерогенных вычислительных системах, использующих графические процессоры, а также численные методы и алгоритмы решения задачи молекулярного докинга.

Предметом исследованияявляется эффективная реализация численных методов оптимизации с учетом особенностей взаимодействия информационных процессов, протекающих в гетерогенных вычислительных системах, использующих графические процессоры.

Научные положения, выносимые на защиту.

1. Алгоритм реализации метода дифференциальной эволюции, учитывающий особенности графических процессоров.

2. Алгоритм расчёта силовых полей межмолекулярного взаимодействия на основе вычисления сеток, адаптированный для графических процессоров.

3. Алгоритм решения задачи молекулярного докинга с использованием графических процессоров.

Научная новизна представленных результатов.

1. Предложен алгоритм реализации метода дифференциальной эволюции на графических процессорах. В отличие от существующих подходов алгоритм использует одноблочную параллельную декомпозицию процедуры оптимизации и выполняет все основные этапы метода дифференциальной эволюции без управления со стороны центрального процессора. Это позволило увеличить количество одновременно выполняемых процедур оптимизации на графическом

процессоре и автоматически управлять их выполнением за счёт учёта особенностей обрабатываемых данных, архитектуры графического процессора и метода вычислений.

2. Предложен алгоритм вычисления сеток межмолекулярного взаимодействия с использованием графического процессора. В отличие от существующих подходов алгоритм унифицировано выполняет вычисление для различных компонент энергии межмолекулярного взаимодействия и использует параллельное управление подготовкой данныхдля вычислений на графических процессорах. Это позволило равномерно распределять нагрузку по всем доступным ресурсам графического процессора и увеличить производительность по сравнению с существующими решениями.

3. Предложен алгоритм решения задачи молекулярного лиганд-белкового докинга на графических процессорах, использующих разработанный подход к выполнению метода дифференциальной эволюции. В отличие существующих подходов алгоритм может выполнять молекулярный лиганд-белковый докинг для нескольких пар химических соединений одновременно на одном графическом процессоре, что позволяет повысить эффективность использования ресурсов графических процессоров и увеличить скорость обработки больших наборов лигандов.

Предложенные алгоритмы создают научную основу решения важных прикладных задач фармакологии и биоинформатики на базе гетерогенных вычислительных систем,использующих графические процессоры.

Практическая значимость полученных результатов.

Разработанные в диссертации алгоритмы реализованы в виде комплекса программ. Применение разработанного комплекса программного обеспечения для решения задача молекулярного лиганд-белкового докинга продемонстрировало существенное увеличение производительности по сравнению с аналогичными решениями. Алгоритм реализации метода дифференциальной эволюции на графических процессорах демонстрирует ускорение до 86 разпри полной загрузке графического процессора по сравнению с решениями, использующими только

возможности центрального процессора. Предложенный алгоритм вычисления сеток межмолекулярного взаимодействия работает до 48 раз быстрее по сравнению с существующими методами решения. При выполнении молекулярного докинга достигается ускорение до 17 раз при использовании одного графического процессора, и до 27 раз при использовании массива графических процессоров.

Апробация работы.

Основные результаты исследования были представлены и обсуждены на следующих конференциях:

1. XII Международная научная конференция «Интеллект и наука», Железногорск, 2012 год.

2. X Международная научно-практическая конференция студентов, аспирантов и молодых учёных «Молодёжь и современные информационные технологии», Томск, 2012 год.

3. Всероссийская научно-практическая конференция «Многоядерные процессоры, параллельное программирование, ПЛИС, системы обработки сигналов», Барнаул, 2013 год.

4. XIIIМеждународная научная конференция «Интеллект и наука», Железногорск, 2013 год.

Результаты диссертации опубликованы в 8 работах, из них в изданиях, рекомендованных Высшей Аттестационной Комиссией – 3 [76, 84, 85].

Получено два свидетельства о регистрации программы для ЭВМ в Федеральном институте промышленной собственности (№2013660687; №2014619771).

Получен акт о внедрении результатов диссертации в ООО «Мобилфон». Рекомендации, содержащиеся в диссертации, использовались для решения задач оптимизации планирования нагруженных вычислений в многосерверной среде. Получен акт о внедрении результатов диссертации в ООО «Функциональные наносистемы». Рекомендации, содержащиеся в диссертации, использовались для решения задачи расчёта кинетики реакции и транспорта на границе раздела

«металлический микросетчатый электрод – органическая электрохромная композиция».

Личный вклад автора.

Результаты, представленные в диссертации, получены непосредственно автором: алгоритм реализации метода дифференциальной эволюции на графических процессорах; алгоритм вычисления сеток силовых полей межмолекулярного взаимодействия на графических процессорах; алгоритм выполнения молекулярного лиганд-бедкового докинга на графических процессорах; комплекс программного обеспечения реализующий указанные алгоритмы. Автором написаны и подготовлены к публикации все статьи из списка работ, представляющих результаты диссертации.

Структура и объём диссертации.

Диссертация включает в себя введение, четыре главы, заключение, список использованных источников, список терминов и определений, приложение. Содержит 23 рисунка, 11 таблиц и 53 формулы. Список использованных источников содержит 135 наименований. Общий объём диссертации – 114 страниц.

В первой главепроблема молекулярного докинга сформулирована как многомерная оптимизационная задача, в которой требуется найти глобальный минимум. В главе проанализированы существующие численные методы оптимизации, применяемые как для локальной, так и для глобальной оптимизации. Исследованы существующие подходы к выполнению молекулярного лиганд-белкого докинга и его основные этапы (перебор конформаций и их оценка). Проведена классификация исследованных подходов к выполнению молекулярного лиганд-белкового докинга. Детально разобраны используемые алгоритмы и их реализации на различных аппаратных платформах, в том числе с использованием гетерогенных вычислительных систем. Исследованы существующие подходы к оценке конформаций молекулярных соединений, а также существующие силовые поля, применяемые для параметризации оценочных функций. Исследованы существующие подходы применения

сеточного метода к молекулярному докингу. Сформулированы требования к методам оптимизации для их эффективной реализации на гетерогенных вычислительных системах с использованием графических процессоров. На основании предъявленных требований в качестве наиболее перспективного численного метода оптимизации для реализации на графических процессорах выбран метод дифференциальной эволюции. Проведённый анализ программно-аппаратных особенностей целевой платформы позволил приступить к разработке алгоритмов, обеспечивающих адаптацию метода дифференциальной эволюции к специализированным вычислительным системам на базе графических процессоров.

Во второй главе проанализированы существующие методывыполнения метода дифференциальной эволюции с использованием графических процессоров, на основе чегопредложен алгоритм, обеспечивающий выполнение дифференциальной эволюции с использованием специализированной вычислительной системы на основе графических процессоров. Предложенныйалгоритм позволил эффективно использовать ресурсы графического процессора за счёт увеличения количества одновременно выполняемых процедур оптимизации на одном графическом процессоре, что было невозможно при использовании предшествующих моделей.

В третьей главе представлен алгоритм вычисления сеток силовых полей с использованием графических процессоров, позволяющий эффективно автоматически распределять вычисления набора сеток силовых полей для большого количества биомишений как для одного графического процессора, так и для массива графических процессоров. Предложен менеджер времени выполнения, обеспечивающий максимальную нагрузку для всех доступных графических процессоров. Представленалгоритм выполнения молекулярного лиганд-белкового докинга с использованием графических процессоров, позволяющий быстро выполнять скрининг больших баз данных лигандов за счёт обработки нескольких лигандов одновременно. Описаны алгоритмы и приёмы, применяемые при выполнении реализации.

В четвёртой главе приведены результаты тестирования предложенных программ выполнения метода дифференциальной эволюции, вычисления сеток силовых полей и выполнения молекулярного лиганд-белкового докинга с использованием различных графических процессоров и их массивов. Использование разработанных моделей и методов позволило повысить эффективность выполнения метода дифференциальной эволюции, вычисления сеток силовых полей и молекулярного лиганд-белкового докинга за счёт учёта особенностей информационных структур и взаимодействия различных информационных процессов и их адаптации к особенностям специализированных вычислительных систем.

Взаключении изложены основные результаты диссертации.

Приложение А содержит акты о внедрении результатов диссертации.

1 Особенности реализации численных методов оптимизации на графических процессорах для решения прикладных задач

Для подбора перспективного численного метода оптимизации, подходящего для решения прикладных задач на графических процессорах,необходимо сформулировать решаемую оптимизационную задачу, выявить особенности программно-аппаратной архитектуры специализированных вычислительных систем и сформулировать ряд дополнительных требований, предъявляемых к данным методам со стороны графических процессоров. После чегонеобходимо рассмотреть особенности реализации различных численных методов оптимизации для решения прикладной задачи молекулярного докинга.

1.1 Виды молекулярного докинга

Молекулярный докинг – это компьютерное моделирование взаимодействия молекул. Основной целью выполнения молекулярного докинга является поиск реалистичных конформаций молекулярных соединений на основе количественной оценки оптимальности их взаимного пространственного расположения, структурной комплементарности и энергии связывания. Молекулярный докинг, как правило, выполняется на паре молекулярных соединений. Выделяют лиганд-белковый и белок-белковый докинг [1,2,50,53].

Процесс выполнения молекулярного докинга довольно условно можно разделить на две стадии: первая – стыковка соединений (*docking*), вторая – оценка образованной конформации (*scoring*). В зависимости от реализации, особенности и специфики алгоритмов, эмпирических соображений разработчиков процесс вычислений может сильно изменяться. В общем случае на вход процедуры подаются трёхмерные структуры молекул, на выходе же необходимо получить некоторое, наиболее выгодное пространственное расположение молекул друг относительно друга, а также некоторую количественную оценку образованного соединения. В ряде случаев на вход может подаваться дополнительная

информация различного рода. Как правило, это координаты области, в которой следует искать наилучшее место соединения.

Трёхмерные структуры молекулярных соединений получают при помощи методов ядерно-магнитного резонанса (ЯМР) и рентгенографии. Значительное количество соединений, представляющих ценность для научного сообщества, опубликовано в открытом доступе [11-19]. Подобные базы данных содержат достаточно подробную информацию как о самой структуре, так и о деталях эксперимента, в процессе которого она была получена.Структуры сопровождаются информацией о соединениях, с которыми они взаимодействуют. Помимо этого, в свете высокой стоимости процедур рентгенографии и ЯМР трёхмерные структуры молекулярных соединений могут предсказываться на основании уже известных трёхмерных структур (полученных точными способами), например, при помощи метода QSAR [20].

Молекулярный докинг имеет значительное количество особенностей, на основании которых может быть классифицирован, но прежде всего, следует различать молекулярный докинг по типам молекулярных соединений, участвующих в процессе, так как этот фактор во многом является определяющим при выборе алгоритмов, которые используются при выполнении моделирования. На основании типов соединений разделяют: белок-белковый и лиганд-белковый молекулярный докинг.

Белок-белковый докинг[2, 5, 6, 23, 37, 38]подразумевает под собой моделирование взаимодействия нескольких белков и применяется для понимания процессов, протекающих в организме. Численные методы оптимизации, как правило, не используются для выполнения белок-белкового докинга, так как основным параметром, характеризующим успешность данного вида докинга, является структурная комплементарность, для анализа которой успешно применяется преобразование Фурье.

Целью **лиганд-белкового докинга** [1,2,25-30,32,35,36,39,42,47-53]является моделирование взаимодействия большой молекулы белка (несколько сотен

атомов) и сравнимо малой молекулы (до нескольких десятков атомов), называемой лигандом.

Модель молекулярного распознавания оказывает прямое влияние на процедуру докинга. Под моделью молекулярного распознавания подразумевается то, какие ограничения налагаются на взаимодействие молекулярных соединений друг с другом в процессе химической реакции. Первая модель, определяющая принципы молекулярного распознавания, была предложена в 1894 г. Фишером (Hermann Emil Fischer) и получила название модель «LockandKey», согласно которой молекула биомишени подходит к молекуле лиганда в случае, если они структурно комплементарны, т.е. подходят друг к другу, как ключ к замку. В настоящее время используется уточнённая модель «inducedfit», предложенная Кошландом (Daniel Koshland) [21, 22]. Исходя из данной модели, молекулы не взаимодействуют как фиксированные структуры и в процессе взаимодействия могут претерпевать конформационные изменения. В частности, данная модель подразумевает ситуацию, когда некоторые боковые цепи молекулы белка смещаются таким образом, что образуется новая, более энергетически эффективная область для взаимодействия с молекулой лиганда. Модель «inducedfit» на данный момент является основой для выполнения гибкого докинга.

Несмотря на то, что была показана подвижность молекул, участвующих во взаимодействии, моделирование этого процесса «как есть», т.е. с учётом всех известных законов, является ресурсоёмкой процедурой,применяемой на практике крайне редко [53]. Как следствие, для ускорения моделирования применяются некоторые способы аппроксимации и упрощения. Одним из вариантов такого упрощения является различный учёт подвижности молекул, что во многом определяет классы алгоритмов, использующиеся при выполнении моделирования [23-29], а также существенно влияет на скорость вычислений и точность результата.

На основании учёта подвижность можно выделить:

– **Фиксированный докинг.** Вид докинга, не учитывающий подвижность молекул, участвующих во взаимодействии. Фиксированный докинг, как правило,

не применяется для расчёта лиганд-белкового взаимодействия и в основном используется для расчёта белок-белковых соединений [23-25].

– **Гибкий докинг.** В свою очередь, данный вид докинга может подразделяться на несколько подвидов в зависимости от учёта подвижности. Как правило, в расчёт берётся либо только подвижность лиганда [26-29], либо дополнительно подвижность некоторых цепочек биомишени [30-32].Полная подвижность биомишени во внимание принимается редко [33-36] в свете высоких вычислительных затрат.

Следует отметить, что выполнение молекулярного лиганд-белкового докинга не является устоявшейся и шаблонной процедурой, несмотря на значительный период времени, в течение которого выполняются исследования в этой области. В целом это можно объяснить разнообразием целей, которые преследуются при выполнении докинга, что существенным образом сказывается на требованиях к производительности и точности конкретного решения. Подавляющее большинство современных программ учитывает подвижность молекулы лиганда и в некоторых случаях подвижность (как правило, частично) молекулы биомишени [30, 35, 36]. При выполнении поиска оптимальной конформации на стадии стыковки выполняется перебор различных пространственных расположений молекул, участвующих в докинге. Даже при моделировании неподвижного лиганда такая задача включает в себя 6 степеней свободы (три степени вращения, три степени смещения), что уже является ресурсоёмкой процедурой. При учёте внутренних степеней свободы молекулы лиганда (торсионных углов) количество степеней свободы может возрастать значительно в зависимости от конкретного лиганда. Исчерпывающий поиск в такой ситуации становится невозможен за разумный период времени. В связи с этим в программах лиганд-белкового докинга активно применяются методы оптимизации[25-30, 39-41, 47-51]. При этом часто процесс оптимизации может быть разделён на два этапа: быстрая и неточная глобальная оптимизация, целью которой является определение некоторого приблизительного позиционирования лиганда в активном сайте (сайте связывания, *bindingsite*) биомишени; медленная

локальная оптимизация для точного позиционирования лиганда в активном сайте биомишени.

Наиболее перспективным для разработки высокопроизводительных алгоритмов для графических процессоров является гибкий лиганд-белковый докинг с учётом подвижности лигандов, ориентированный на обработку больших баз химических соединений. Ускорение и повышение точности данного вида докинга позволит существенным образом увеличить скорость и повысить качество подбора перспективных химических соединений при разработке лекарственных препаратов. Данный вид докинга требует обработки большого объёма данных, в свете чего привлечение значительных параллельных ресурсов графических процессоров позволит обеспечить существенный прирост производительности.

1.2 Оптимизируемая функция. Оценкаэнергии конформации

В качестве оптимизируемой функции в большинстве программ выполнения докинга независимо от применяемого метода численной оптимизации, как правило, используется некоторый вариант функции оценки энергии связывания (*bindingenergy*) конформации, которая позволяет определить качество образованной конформации, т.е. ответить на вопрос о принципиальной возможности протекания такой реакции, а также выбрать, какая из реакций энергетически более предпочтительна. Энергия связывания должна быть оценена исходя из трёх компонентов: свободной энергии лиганда без биомишени,свободной энергии биомишени без лиганда, свободной энергии образованной конформации (1.1). При этом основной целью является поиск минимума энергии.

$$G_{bind}=G_{complex}-(G_{target}+G_{ligand}) \quad (1.1)$$

В целях упрощения оптимизации ряд компонентов может быть исключен из вычислений. Так, при выполнении лигад-белкового докинга с учётом подвижности только лиганда из рассмотрения может быть исключена свободная

энергия биомишени, так как биомишень не претерпевает изменений в процессе докинга, следовательно, для выполнения оптимизации её энергия не обязательна.

На практике для выполнения докинга достаточно оценить только энергию образованной конформации и часть энергии лиганда. Согласно методам квантовой механики (молекулярно-механический подход), энергия может быть представлена как линейная комбинация нескольких компонент [2, 53-61]:

$$E_{total} = E_{intro} + E_{inter}, \quad (1.2)$$

где E_{intro} – энергия внутримолекулярного взаимодействия, E_{inter} – энергия межмолекулярного взаимодействия. В свою очередь, энергия внутримолекулярного взаимодействия может быть представлена как линейная комбинация следующих компонент:

$$E_{intro} = E_{bond} + E_{angle} + E_{torsion}, \quad (1.3)$$

где:

– E_{bond} – энергия взаимодействия между атомами, разделёнными одной ковалентной связью (атомы 5 и 3 на рисунке 1.1);

– E_{angle} – энергия взаимодействия между тремя ковалентно-связанными атомами (атому 1, 2 и 3 на рисунке 1.1);

– $E_{torsion}$ – энергия взаимодействия между атомами, разделёнными тремя ковалентными связями, образующими двугранный (торсионный) угол (атомы 1, 2, 3 и 4 на рисунке 1.1).

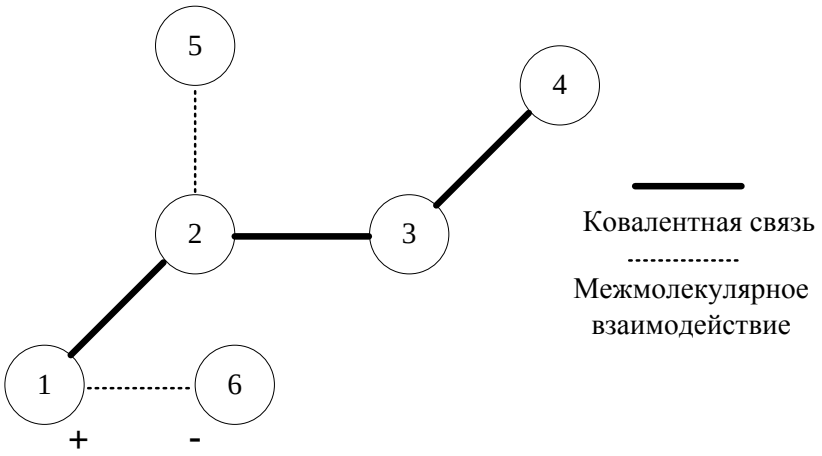


Рисунок 1.1 - Модель атомов молекулы

Кроме того, иногда выполняется оценка *improper*двугранных углов, т.е. углов, образованных четырьмя соединёнными ковалентными связями атомами, но при этом не образующих правильный двугранный угол (рисунок 1.2).

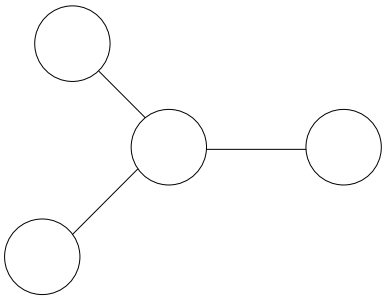


Рисунок 1.2 - Improperдвугранный угол

Было отмечено, что ряд подобных углов имеет устойчивые пространственные расположения [59,62, 63].

Энергия межмолекулярного взаимодействия может быть представлена как линейная комбинация следующих компонент:

$$E_{inter} = E_{vdw} + E_{elec}, \quad (1.4)$$

где:

– E_{vdw}– энергия Ван-дер-Ваальсового взаимодействия (атомы 2 и 5 на рисунке 1.1);

– E_{elec}– энергия электростатического взаимодействия между заряженными атомами (атомы 1 и 6 на рисунке 1.1).

Также некоторые программы могут отдельно оценивать энергию водородных связей. Межмолекулярное взаимодействие вносит основной вклад в энергию связывания.

Энергию Ван-дер-Ваальсового взаимодействия рассчитывают на основе потенциала Леннарда-Джонсона:

$$E_{vdw} = \sum_{i<j} \varepsilon_{ij} \left[\left(\frac{r_{0ij}}{r_{ij}} \right)^{12} - 2 \left(\frac{r_{0ij}}{r_{ij}} \right)^{6} \right], \quad (1.5)$$

где r_{ij} – расстояние между взаимодействующими атомами, r_{0ij}– сумма Ван-дер-Ваальсовых атомов взаимодействующих атомов,ε_{ij} – размерность энергии взаимодействующих атомов. Потенциал Леннарда-Джонсона отражает парное

притяжения между атомами, находящимися на расстоянии больше суммы их Ван-дер-Ваальсовых радиусов в терме 6, а также парное отталкивание в случае, если атомы находятся на расстоянии меньше суммы Ван-дер-Ваальсовых атомов в терме 12. Часто потенциал Леннарда-Джонсона называются потенциалом 6-12.

Энергию электростатического взаимодействия принято рассчитывать, базируясь на законе Кулона:

$$E_{elec} = 332 \sum_{i<j} \frac{q_i q_j}{\varepsilon r_{ij}} \quad (1.6)$$

где q_i, q_j – заряды взаимодействующих атомов,r_{ij} – расстояние между взаимодействующими атомами,ε– диэлектрическая проницаемость среды, 332 – переводной коэффициент в ккал/моль.

Компоненты энергии внутримолекулярного взаимодействия E_{bond},E_{angle},$E_{torsion}$являются штрафными функциями (*penalty functions*), т.е. отражают влияние отклонения структуры от идеальной. В идеале суммарный вклад данных компонент должен стремиться к нулю.

Энергия взаимодействия между атомами, разделёнными одной ковалентной связью (E_{bond}), рассчитывается по формуле:

$$E_{bond} = \sum_{bonds} (b - b_0)^2, \quad (1.7)$$

где b – длина ковалентной связи для некоторой пары атомов, b_0 – «идеальная» длинна ковалентной связи.

Энергия взаимодействия между тремя ковалентно-связанными атомами (E_{angle}):

$$E_{angle} = \sum_{angles} (\theta - \theta_0)^2, \quad (1.8)$$

где θ – валентный угол для некоторой группы из трёх атомов, θ_0 – «идеальный» валентный угол.

Энергия взаимодействия между атомами, разделёнными тремя ковалентными связями ($E_{torsion}$), вычисляется по формуле:

$$E_{torsion} = \sum_{torsions} (1 + \cos(n\chi - \delta)), \quad (1.9)$$

где χ – торсионный угол; n – периодичность торсионного барьера; δ – фазовый сдвиг.

Выполнение оценки конформации является чрезвычайно вычислительно затратной процедурой, поэтому при реализации конкретной оценочной функции часто применяются различного рода аппроксимации и упрощения с целью ускорения процедуры. Кроме того, очень часто при выполнении молекулярного докинга не стоит задача осуществить максимально приближенное к реальности моделирование, а необходимо произвести максимально быструю обработку (*screening*) некоторой большой базы лигандов с целью отобрать из нескольких миллионов вариантов несколько десятков или сотен для дальнейшего более точного моделирования. Соответственно, при выполнении такой процедуры нет необходимости добиваться максимальной точности в оценки энергии.

Корректная оценка энергии невозможна без правильно подобранных параметров, использующихся в формулах для расчета компонентов энергии. Совокупность таких параметров называется силовым полем. Силовые поля различаются прежде всего механизмами типизации атомов молекул и, собственно, значениями параметров, которые характеризуют трёхмерные структуры.

Существует значительное количество силовых полей и их различных модификаций. Наиболее популярными и общеупотребительными наборами силовых полей являются AMBER [54, 62, 71, 72], CHARMM [55, 63], MMFF [64], ММ [65-67]. Указанные силовые поля ориентированы на различные типы химических соединений: нуклеиновые кислоты, белки, небольшие молекулы.

1.3 Численные методы оптимизации

В рамках прикладной задачи рассматриваются численные методы оптимизации, которые применяются в существующих решениях задачи молекулярного лиганд-белкового докинга: генетический алгоритм, метод Монте-Карло, муравьиный алгоритм. Помимо этого,анализируется возможность использования ряда других стохастических методов оптимизации.

1.3.1 Генетический алгоритм в программеAutoDock

Одной из самых распространенных и точных программ для выполнения лиганд-белкового докинга является программа AutoDock, разработанная в Исследовательском Институте Скрипсса (The Scripps Research Institute). Программа использует генетический алгоритм,который относится к методам стохастического поиска для выполнения глобальной оптимизации [27, 30, 39].

Согласно данному алгоритму оптимизируемые переменные представляются как вектор генов, называемый хромосомой. Применительно к AutoDock используются три переменные для определения смещения лиганда и четыре переменные для кватернионов. Помимо такого подхода, могут явно задаваться Эйлеровы углы вращения вокруг координатных осей вместо кватернионов, как это делается при реализации генетического алгоритма в программе SOL [40, 41]. Кроме того, в хромосоме дополнительно отводится по переменной для каждого торсионного угла лиганда, который будет изменяться в процессе моделирования.

Каждый шаг генетического алгоритма состоит из нескольких этапов: вычисление функции фитнеса, отбор (селекция), кроссовер (кроссинговер, *crossover*), мутация и отбор элиты. AutoDock использует двухточечный кроссинговер, который выполняется между парами хромосом. Программа SOL использует одноточечный кроссинговер. Далее гены в хромосоме подвергаются мутации. В AutoDock мутация определяется некоторой добавкой к гену, которая задаётся распределением Коши. После выполнения мутации выбирается набор хромосом, имеющий самые лучшие значения фитнес. Эти хромосомы называются элитой и переносятся на следующую итерацию алгоритма без изменений.

AutoDock поддерживает моделирование подвижности отдельных цепей биомишени. Для этого пользователем отбираются те цепи, которые будут рассматриваться как подвижные, после чего они моделируются явно, по аналогии с моделированием движения лиганда.

Помимо указанных выше AutoDock и SOL, генетический алгоритм реализован в ряде других программ [32, 42].

Генетический алгоритм содержит в себе комплексные правила для отбора жизнеспособной популяции и имеет достаточно сложный внутренний этап кроссинговера (особенно если применяется двухточечных кроссинговер). Отсутствие фиксированного размера популяции, а также наличие различных правил по формированию популяции на каждом этапе является существенным ограничением, так как на графических процессорах в зависимости от модели либо отсутствует динамическое выделение памяти, либо оно требует существенных накладных расходов. Выделение же пула памяти с достаточным запасом может привести как к необоснованным накладным расходам в ситуациях, когда с течением алгоритма не требуется наличие большого количества векторов приближенных решений, так и к искусственному ограничению работы алгоритма из-за необходимости исключать потенциально значимые приближенные решения по причине жёстко заданного максимального объёма памяти. Существующие попытки ускорения с использованием графических процессоров программы AutoDock показывают, что распараллеливание отдельных этапов алгоритма, в частности мутирования и кроссинговера, не приносит существенного повышения производительности [43, 44].

1.3.2 Итеративный локальный поиск в программе AutoDock Vina

Одной из самых быстрых существующих программ для выполнения молекуляного докинга на данный момент является AutoDockVina [26]. Программа также была разработана в Исследовательском Институте Скрипсса. Заложенный алгоритм использует метод Итеративного поиска [45] в качестве процедуры глобального поиска в сочетании с алгоритмом Бройдена-Флетчера-Гольдфарба-Шанно (*Broyden-Fletcher-Goldfarb-Shanno, BFGS*) как локальный поиск [46].Алгоритм случайным образом инициализирует приближенное решение, после чего итеративно применяет операции локального поиска (внесение изменений в приближенное решение, а также отбор).

По результатам тестов, версия AutoDockVina, использующая один процессор, демонстрирует прирост производительности по сравнению с AutoDock в 62 раза. При этом версия, использующая 8-ядерный процессор, демонстрирует ускорение в 7,25 раз по сравнению с однопроцессорной версией [26].

1.3.3 Метод инкрементального наращивания в программе DOCK

Алгоритм DOCK использует идею метода якорно-инкрементального наращивания (*anchor-and-growmethod*) [25, 28, 47, 48]. В соответствии с этим методом лиганд разделяется на несколько небольших частей, которые полагаются неподвижными в процессе моделирования.

Разделение лиганда на фрагменты выполняется согласно торсионным углам. Кольцевые структуры полагаются неподвижными. Из множества фрагментов отбирается одна, наиболее крупная, которая считается якорем (*anchor*), и помещается в сайт связывания биомишени. Остальные части лиганда разделяются на уровни согласно количеству вращающихся связей, отделяющих их от якоря. Далее идёт итеративное наращивание лиганда. К якорю добавляются фрагменты, которые могут быть различным образом позиционированы согласно вращающимся связям. В результате получается некоторый набор структур, который подвергается обрезке (*pruning*). Для этого для каждой структуры производятся оценки энергии взаимодействия, на основе которых отбирается эталонная. После чего при помощи вычисления взвешенного среднеквадратичного отклонения от эталонной структуры определяются те, которые будут исключены из дальнейшего моделирования. Взвешенное среднеквадратичное отклонение вычисляется по формуле:

$$wRMSD = \left(\frac{\sum_{i=1}^{N} L_i \left[\left(x_i^c - x_i^r\right)^2 + \left(y_i^c - y_i^r\right)^2 + \left(z_i^c - z_i^r\right)^2 \right]}{\sum_{i=1}^{N} L_i} \right)^{\frac{1}{2}}, \quad (1.10)$$

где (x^c, y^c ,z^c)– координаты атома кандидата; (x^r, y^r, z^r) – координаты атомов эталонной конформации; L_i– уровень, на котором находится атом; N – количество атомов. Система весов, учитывающая уровни, добавлена по причине того, что атомы, находящиеся на более высоких уровнях, оказывают большее влияние на итоговую конформацию.

В процессе наращивания, а также после формирования итогового набора конформаций выполняется локальный поиск. Алгоритм DOCK для этих целей использует симплексный метод [46].

К недостаткам метода следует отнести нерегулярность структуры данных. Итеративное изменение набора фрагментов потребует перераспределения их в памяти, а также динамического выделение памяти, что является медленной операцией на графическом процессоре. В совокупности это приведёт как к замедлению работы с памятью, так и увеличению количества обращений к ней из-за нарушения условий для объединения запросов.

1.3.4 Метод Монте-Карло в программе ROSETTALIGAND

Метод Монте-Карло также используется для выполнения оптимизации в нескольких программах докинга [29, 35, 36, 49, 50]. Программа ROSETTALIGAND, помимо учёта подвижности лиганда, принимает во внимание подвижность цепочек биомишени. При этом цепочки определяются не пользователем, а на основании существующих баз данных ротамеров.

После случайной инциализации используемый в программе протокол Монте-Карло итеративно проходит три основных этапа:

- случайное изменение позиции лиганда на 0,1 ангстрем по координатным осям, а также случайный поворот вокруг координатных осей на 0,05°;
- внесение изменений в цепочки биомишени;
- внесение изменений в торсионные углы лиганда и выполнение локальной оптимизации.

В качестве метода локальной оптимизации используется алгоритм Давидона-Флетчера-Пауэлл (*Davidon–Fletcher–Powell, DFP*), который очень похож на алгоритм BFGS [46]. После внесения изменений выполняется подсчёт энергии конформации. Закрепление изменений определяется посредством критерия Метрополиса.

Метод Монте-Карло в общем случае демонстрирует достаточную низкую скорость сходимости по сравнению с альтернативными методами. Помимо этого, применение метода оправдано в случаях, когда предметная область не описывается аналитически, что несправедливо для решаемой прикладной задачи.

1.3.5 Муравьиный алгоритм в программе PLANTS

ПрограммаPLANTS [51] используетмуравьиныйалгоритмдляоптимизациидокинга, аименноMax-MinAntSystem (*MMAS*),версиюалгоритма [113].

Согласно алгоритму, вектор приближенного решения ассоциируется с каждой степенью свободы лиганда и имеет размерность, соответствующую всем значениям, которые может принимать величина. Так, для смещения по координатным осям задан шаг в 0,1 ангстрем. Для поворотов как по валентным углам, так и по торсионным задан шаг в 1°. Таким образом, например, вектор для вращения вокруг оси X будет иметь размерность 360. Тропа (в терминах алгоритма) отражает привлекательность (количество феромонов) значения *j* для переменной *i*.В качестве локального поиска программа использует симплексный метод.

К недостаткам муравьиного алгоритма следует отнести математический аппарат, содержащий значительное количество ветвлений, что может привести к дивергентным варпам и исчерпанию регистрового файла.Существующая попытка ускорения муравьиного алгоритма на основе алгоритма PLANTS показывает, что

искусственное увеличение количества приближенных решений с целью загрузки ресурсов графического процессора приводит к увеличению обмена информацией между приближенными решениями, т.е. уменьшает общее время параллельной работы алгоритма [52].

1.3.6 Метод дифференциальной эволюции

Метод дифференциальной эволюции впервые предложен в [77]. Решение задачи оптимизации некоторой целевой функции определяется наборомвекторов, который называется кластером, или популяцией. Каждый элемент вектора – это параметр, участвующий в процедуре оптимизации. Алгоритм подразумевает три основные стадии: мутация, кроссинговер, селекция. Формирование набора мутантных векторов выполняется на каждой итерации метода. За основу берутся вектора из предыдущей итерации.После формирования мутантного вектора с его использованием происходит формирование тестового (*trial*) вектора. Для каждого тестового вектора из набора выполняется вычисление целевой функции. В случае если значение целевой функции ближе к экстремуму, то тестовый вектор замещает соответствующий ему вектор кластера.

1.3.7 Метод роя частиц

Метод роя частиц[126] в целом достаточно похож на эволюционные алгоритмы в плане организации внутренних структур. При этом основным внутренним этапом, влияющим на значения векторов приближенных решений, является изменение положения частицы (вектора приближенных решений) в пространстве на основании некоторого вектора скорости, ассоциированного с частицей.

К недостаткам метода роя частиц следует отнести необходимость поддержания общего состояния роя, что потребует в некоторый момент времени произвести редукцию результатов вычислений по всем нитям графического

процессора. Выполнение редукции так или иначе приведёт к простою нитей графического процессора. Помимо этого, метод требует двойного набора структур данных: один набор для приближенного решения (частиц), второй набор для скорости их движения.

1.4 Требования к численным методам оптимизации для реализации с использованием графических процессоров

На текущий момент производством графических процессоров, поддерживающих вычисления общего назначения, занимаются компании AMD и NVIDIA. Графические процессоры AMDне предоставляют самостоятельной программной платформы, а используют открытый стандарт программирования для гетерогенных вычислительных систем OpenCL [104-108]. При этом доминирующее положение на ранке занимают графические процессоры компании NVIDIA и её программно-аппаратная платформа CUDA[86-103]. Исходя из списка 500 самых мощных суперкомпьютеров, почти все системы, использующие графические процессоры, комплектуются видеокартами от NVIDIA. Одним из важных дополнительных преимуществ NVIDIA является наличие большого набора пакетов и библиотек, в значительной степени упрощающих разработку программного обеспечения и предлагающих готовые решения для ряда устоявшихся задач.Поэтому платформа CUDA выбрана в качестве целевой для проведения исследований.

Графические процессоры в силу особенностей аппаратной архитектуры и программной парадигмы в значительной степени отличаются от центральных процессоров. Их анализ позволил сформировать ряд специфичных требований, предъявляемых к численным методам оптимизации (помимо скорости сходимости и точности) для их эффективной реализации на графических процессорах.

Метод должен иметь высокую степень параллелизма, направленную на повышение точности алгоритма. Основной концептуальной особенностью,

отличающей типовой графический процессор от центрального, является наличие большого количества достаточно простых вычислительных ядер. Такая организация обусловлена тем, что при обработке графики в большинстве случаев необходимо произвести некоторую операцию над заданным набор точек в трёхмерном пространстве, причём эта операция может быть осуществлена независимо над каждой точкой. Таким образом, максимизация простых вычислителей напрямую влияет на скорость обработки графической информации.

В данной платформе каждый такой вычислитель называется CUDA ядро (*CUDA core*). Ядра комплектуются в наборы, которые в сочетании со вспомогательным аппаратным обеспечением образуют мультипроцессоры. Графический процессор комплектуется несколькими мультипроцессорами, как правило, их число не превосходит 16 для современных графических процессоров, ориентированных на научные расчеты [86-89,92]. Схематично графический процессор представлен на рисунке 1.3.

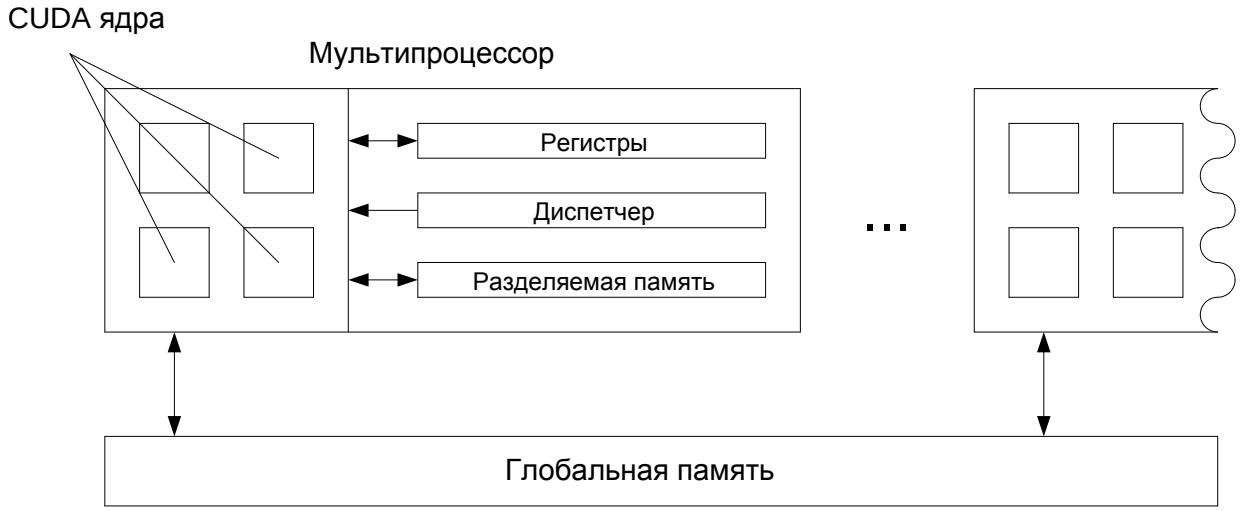


Рисунок 1.3 - Схематичное представление архитектуры графического процессора

Основной парадигмой организации вычислений на графических процессорах является «Одна инструкция для всех данных» (Single Instruction Multiple Data, *SIMD*), что обусловлено большим количеством CUDA ядер [86-89, 92]. Таким образом, наиболее удобными задачами для решения на графических процессорах являются задачи, имеющие в себе существенный параллелизм по данным. Следовательно, для эффективной реализации численного метода

оптимизации на графических процессорах его внутренняя идея должна базироваться на однотипной обработке большого количества однотипных структур данных. При этом точность и сходимость метода должны находиться в прямо пропорциональной зависимости от количества обрабатываемых структур (пусть и при наличии некоторой границы).

Вычислительный процесс метода оптимизации должен быть максимально линейным. Фактически минимальной единицей исполнения является не CUDA нить, а группа нитей, называемая варпом (*warp*), которая и исполняется физически параллельно. Каждый мультипроцессор оборудован как минимум одним планировщиком для исполнения варпов. Современные графические процессоры могут иметь до четырёх планировщиков. Особенностью выполнения варпов является необходимость исполнения всеми нитями в пределах одного варпа одной и той же инструкции. В случае если этого не происходит, выполняется линеаризация (такая ситуация также называется дивергенцией варпа) вычислений: сначала выполняется инструкция для одной части варпа, после чего для другой (в случае исполнения двух различных ветвей). В результате линеаризации вычислений в лучшем случае время выполнения удваивается [86-97]. Таким образом, значительное ветвление в алгоритме оптимизации приведёт к существенному росту количества дивергентных варпов и, как следствие, существенному падению производительности по сравнению с центральным процессором.

Метод оптимизации не должен требовать синхронизации между потоками вычислений или должен использовать лишь барьерную синхронизацию в пределах блока CUDA нитей. Организация глобальной синхронизации в пределах блока требует либо синхронного вызова набора вычислительных ядер, либо использования глобальной памяти в сочетании с атомарными операциями. Оба варианта так или иначе вносят задержки в процесс вычисления. Это обусловлено отсутствием встроенных программных средств синхронизации на данном уровне. Платформа CUDA предоставляет средства барьерной синхронизации только на уровне вычислительных блоков.

Графические процессоры имеют многоуровневую иерархию памяти, объём которой обратно пропорционален скорости доступа (рисунок 1.4). К основным типам памяти следует отнести: глобальную память, разделяемую память и регистровый файл [86-87, 89, 92]. Основным хранилищем графического процессора является глобальная память, которая имеет наибольший объём (до нескольких гигабайт) и является самым медленным типом памяти графического процессора. В глобальную память пересылаются все данные для выполнения вычислений из оперативной памяти. Данные в глобальной памяти доступны всем нитям вычислительной сетки и остаются доступными между вызовами вычислительных ядер.

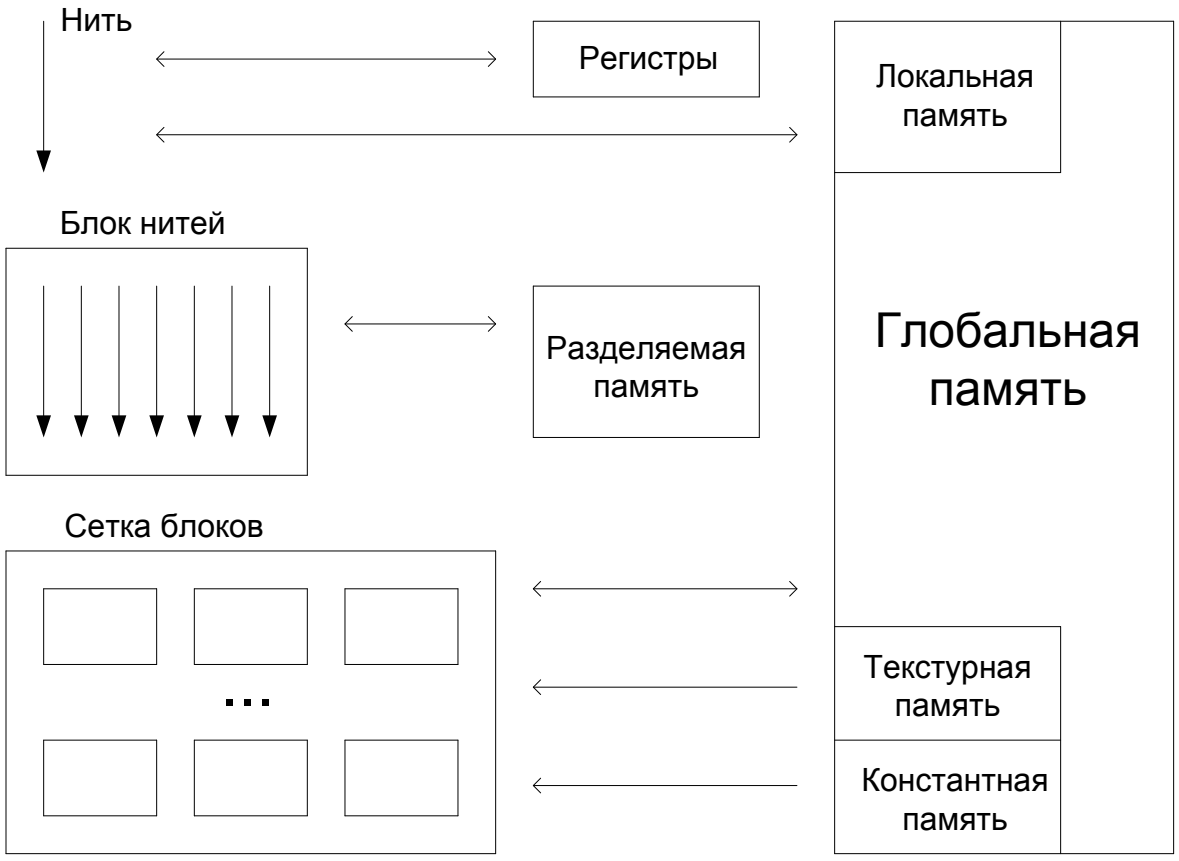


Рисунок 1.4 - Логическая модель вычислений и иерархия памяти платформы CUDA

С целью ускорения обращения к глобальной памяти данные реализуемого алгоритма должны быть организованы таким образом, чтобы запросы к ним в глобальной памяти были объединены аппаратно. Для этого структура данных должна быть регулярной в адресном пространстве глобальной памяти, т.е.

выровненной по адресам так, чтобы каждая нить в пределах варпа выполняла обращения последовательно. Только в таком случае запросы будут объединены. Если обращения к данным при выполнении метода будут нерегулярными, то это приведёт к выполнению отдельного запроса к глобальной памяти для каждой нити в пределах варпа, т.е. увеличит количество запросов до 32 раз.

Минимальное количество входных параметров и этапов метода. Наиболее быстрым и наименее ёмким типом памяти является регистровый файл. Графические процессоры имеют ограничение как на максимальное количество регистров, выделяемое на мультипроцессор, так и количество регистров, выделяемое для одной CUDA нити. Количество регистров, используемое каждой нитью вычислительного ядра, напрямую влияет на количество активных блоков, которое может одновременно обрабатываться на каждом мультипроцессоре (*occupancy*). Программист не имеет средств для управления регистровым файлом при программировании на высокоуровневом языке. Таким образом, на количество используемых регистров можно влиять только опосредовано за счёт применения более экономичного алгоритма. Превышение лимита на количество используемых нитью регистров приводит к размещению информации в локальной памяти графического процессора, которая физически является частью глобальной памяти и, как следствие, имеет низкую скорость доступа.

1.5 Выбор метода оптимизации для реализации на графических процессорах

С точки зрения параллелизма по данным, наибольший интерес для реализации на графических процессорах представляют методы глобальной оптимизации, так как они в большей степени ориентированы на архитектуру графических процессоров — большое количество вычислительных ядер. Методы локальной оптимизации не содержат высокой степени параллелизма по данным, что делает их менее пригодными для выполнения на графических процессорах, по сравнению с методами глобальной оптимизации. Самую высокую степень параллелизма по данным, заложенную в алгоритм, имеют метод

дифференциальной эволюции, генетический алгоритм, а также метод роя частиц. Указанные алгоритмы содержат в себе большой набор однородных данных, представляющих приближенные решения, операции над которыми по ходу выполнения алгоритма могут быть выполнены независимо. При этом генетический алгоритм по сравнению с методом дифференциальной эволюции включает в себя сложный этап отбора жизнеспособной популяции, что подразумевает комбинирование результатов вычислений и, как следствие, ограничивает параллельное выполнение [43,44]. Метод роя частиц, как и метод Монте-Карло, также включают в себя этап, требующий комбинации результатов вычислений. Метод дифференциальной эволюции, в свою очередь, не подразумевает этаповобобщения промежуточных результатов, требует большой набор векторов (вплоть до полной загрузки мультипроцессора в случае многомерной задачи, включающей в себя до 100 оптимизируемых параметров), представляющих собой приближенные решения.При этом каждый вектор может быть обработан независимо в пределах итерации, что хорошо подходит для использования SIMD парадигмы программирования.

С точки зрения максимальной линейности вычислительного процесса, метод дифференциальной эволюции не предполагает значительного ветвления в процессе выполнения. Основными точками принятия решений являются фазы одноточечного кроссинговера, когда происходит случайное изменение параметров приближенного решения, и фаза отбора, когда принимается решение о замещении векторов (при этом обе точки на сам ход алгоритма влияния не оказывают, т.е. не направляют вычисления по различным путям, а только влияют на используемые данные). Генетический алгоритм включает в себя этап двухточечного кроссинговера, реализация которого чревата расхождением варпов и, как следствие, падением производительности. Этап обновления феромона в муравьином алгоритме включает в себя сложное ветвление, которое также приведёт к расхождению варпа.

Генетический алгоритм, включающий в себя этап переноса успешной популяции с одной итерации на другую, требует либо динамического

распределения памяти, что является медленным процессом на графическом процессоре, либо выделение заранее большего объёма памяти, что приведёт к перерасходу памяти по сравнению с другими методами, а также к необходимости перераспределения информации в памяти с целью сохранить объединение запросов к глобальной памяти.

Кроме того, следует отметить, что количество входных параметров метода дифференциальной эволюции ограничено двумя, что не требует существенных дополнительных расходов памяти. Кроме того, алгоритм не требуетзначительного обмена информацией между различными нитями за исключением стадии мутации, использует лишь барьерную синхронизацию в пределах одной итерации, что не может существенным образом ограничить вычислительный процесс. Алгоритм не требует существенных затрат памяти для работы. Фактически для представления основной информации о работе метода необходима память, равная (D+1)*NP*sizeof(float) для каждого набора векторов: текущей итерации, мутантного, тестового. Для сравнения: метод роя частиц требует двойного количества памяти,генетический алгоритм подразумевает неравномерное распределение памяти. При этом с течением алгоритма количество требуемой памяти не изменяется, что важно, так как динамическое выделение памяти на некоторых графических процессорах невозможно, на других же требует чрезвычайно больших накладных расходов.

По результатам анализа, для дальнейшего изучения выбран метод дифференциальной эволюции, так как подход содержит в себе высокую степень параллелизма по данным, требует меньших затрат памяти по сравнению с большинством других алгоритмов, не подразумевает значительного ветвления в процессе выполнения и требует только барьерной синхронизации.Метод наилучшим образом соответствует SIMD парадигме графических процессоров, сохраняет значительный объём как глобальной памяти, так и регистрового файла под решение прикладной задачи и позволяет ускорять работу с медленной глобальной памятью графического процессора за счёт объединения запросов. Кроме того, как следует из результатов сравнительных тестов метода

дифференциальной эволюции и других численных методов оптимизации, представленных в работе [77], метод дифференциальной эволюции демонстрирует превосходство над альтернативными походами с точки зрения скорости сходимости, а также позволяет находить глобальный минимум для оптимизационных задач, содержащий множество локальных минимумов.

1.6 Выводы по первой главе

В главе исследованы существующие виды молекулярного докинга. Наиболее перспективным для дальнейших исследований выбран лиганд-белковый молекулярный докинг. По результатам анализа, выделены основные этапы выполнения молекулярного лиганд-белкового докинга. Задача докинга, сформулирована как оптимизационная проблема поиска глобального минимума энергии взаимодействия молекулярных соединений.Изучены существующие методы численной оптимизации, применяемые для решения задачи молекулярного лиганд-белкового докинга. Исследованы информационные процессы, протекающие в гетерогенных вычислительных системах, на основании чего разработаны требования, предъявляемые к методам численной оптимизации для их эффективной реализации на графических процессорах. На основании разработанных требований для дальнейших исследований выбран метод дифференциальной эволюции,наилучшим образом удовлетворяющий всем предъявленным требованиям, имеющий высокую степень параллелизма по данным и максимально линейный вычислительный процесс.

Результаты, приведённые в главе, опубликованы в работах [3,68].

2 Алгоритмреализации метода дифференциальной эволюции с использованием графических процессоров

В главе рассмотрены существующие подходы к выполнению метода дифференциальной эволюции с использованием графических процессоров. Выявлены их достоинства и недостатки на основание которых предложен новый алгоритм реализация метода дифференциальной эволюции. Рассмотрены особенностипредложенного алгоритма с учётом специфики графических процессоров.

2.1 Существующие подходы к реализации метода дифференциальной эволюции с использованием графических процессоров

Модель выполнения большого количества процедур оптимизации на основании метода дифференциальной эволюции с использованием возможностей графических процессоровможно представить как:

$$M = (DE, GPU), \tag{2.1}$$

$$DE = (de_1, de_2, \dots, de_n), \tag{2.2}$$

$$GPU = (gpu_1, gpu_2, \dots, gpu_m), \tag{2.3}$$

где DE –множество всех процедур оптимизаций, которые необходимо выполнить;GPU–множество доступных графических процессоров. При этом:

$$\forall i \in [1, n], de_i \triangleq (D_i, F_i, CR_i, g_i), \tag{2.4}$$

где D_i – количество оптимизируемых параметров, F_i – константа мутации ($F_i \in [0,2]$), CR_i –константа кроссинговера для i-ой процедуры оптимизации ($CR_i \in [0,1]$), g_i –оптимизируемая функция.Решение задачи оптимизации некоторой целевой функции определяется набором из NP_iвекторов размерностью D_i. Набор векторов называется кластером, или популяцией (V_i).Размерность популяции определяется согласно формуле:

$$|V_i| \triangleq NP_i \in [5D_i, 10D_i] \quad (2.5)$$

Начальное состояние векторов кластера определяется случайным образом в пределах пространства поиска. Алгоритм подразумевает три основные стадии: мутация, кроссинговер, селекция. На каждой последующей итерации метода $(G+1)$ для каждого вектора кластера происходит формирования мутанта согласно формуле:

$$Vm_{k(G+1)} \triangleq V_{r1(G)} + F_i\left(V_{r2(G)} - Vm_{r3(G)}\right), \forall k \in [1, NP_i], \quad (2.6)$$

где $Vm_{k(G+1)}$ – мутантный вектор для k-го вектора; $V_{r1(G)}, V_{r2(G)}, V_{r3(G)}$ – некоторые вектора из предыдущей итерации метода, выбранные случайно.

После формирования мутантного вектора с его использованием происходит формирование тестового вектора согласно формуле:

$$Vt_{kl} \triangleq \begin{cases} Vm_{kl}, rand() \leq CF_i \\ V_{kl}, rand() > CF_i \end{cases}, \forall k \in [1, NP_i], \forall l \in [1, D_i], \quad (2.7)$$

где Vt_{kl} – l-ый параметр k-го тестового вектора, Vm_{kl} – l-ый параметр k-го мутантного вектора, V_{kl} – l-ый параметр k-го вектора текущей итерации метода.Основным вопросом при выполнении моделирования является определение преобразования f такое, что:

$$DE \xrightarrow{f} GPU \quad (2.8)$$

будет наиболее эффективным, т.е. распределение процедур оптимизации по доступным вычислительным ресурсам будет произведено наилучшим образом, с минимальными издержками.

Первая реализация метода дифференциальной эволюции с использованием графических процессоров была представлена в работе [78]. В предложенной работе все стадии алгоритма реализованы посредством отдельного вычислительного ядра, что обеспечивает синхронизацию всех стадий алгоритма. Данные между вызовами ядер располагаются в глобальной памяти графического процессора. Основные управляющие решения по выполнению алгоритма принимаются на центральном процессоре. При этом производится

предварительная генерация случайных чисел в необходимом объёме до вычислений и перенос их в память графического процессора. Реализация использует мультиблочную вычислительную CUDA сетку для организации вычислений. Каждая нить графического процессора обрабатывает отдельный вектор из набора. Схематично процесс представлен на рисунке 2.1.

Алгоритм включает в себя 6 CUDA ядер:

- Ядро I. Ядро выполняет первоначальную инициализацию кластера векторов приближенных решений метода с использованием случайно сгенерированных чисел;
- Ядро E. Ядро вычисляет значение целевой функции для каждого вектора приближенного решения.
- Ядро P. Подготовка к выполнению мутации. В данном ядре происходит вычисление набора индексов векторов, которые будут использоваться для выполнения мутации.
- Ядро M. Выполняется мутирование векторов согласно индексам, вычисленным в ядре P;
- Ядро C. Кроссинговер. Формирование тестовых векторов.
- Ядро R. Отбор. На основании значения целевой функции ядро выполняет замену векторов из предыдущей итерации соответствующими тестовыми векторами.

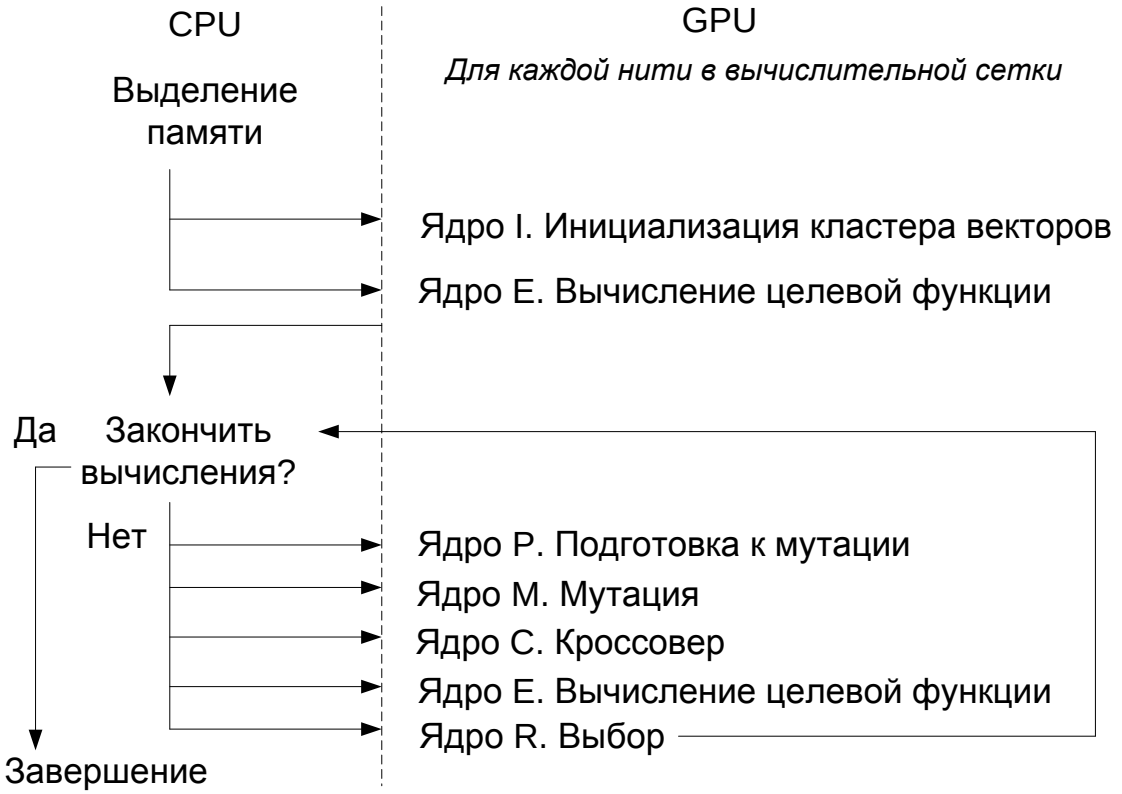


Рисунок 2.1 — Описание реализации метода дифференциальной эволюции на графическом процессоре

Таким образом, данный подход определяет модель как:

$$M_1 = (DE, GPU, CPU), \tag{2.9}$$

где CPU - центральный процессор, выполняющий функцию предварительной генерации случайных чисел. Одна процедура оптимизации, описанная в (2.4), задаётся как:

$$\forall i \in [1, n], de_i = (D_i, F_i, CR_i, g_i, RAND_i), \tag{2.10}$$

где $RAND_i$ – множество предгенерированных случайных чисел, такое, что:

$$\forall p \in [1, |RAND_i|], RAND_{ip} \in \mathbb{R} \tag{2.11}$$

Кроме того, реализация ориентирована только на работу с одним графическим процессором, т.е.:

$$|GPU| = 1, gpu = (blocks, threads), blocks - const, threads - const \tag{2.12}$$

где $blocks$ – количество вычислительных блоков, $threads$ –количество CUDA нитей в вычислительном блоке. Итоговое преобразование в данном случае выглядит следующим образом:

$$f_1 = CPU \rightarrow (de_i \rightarrow gpu), \forall i \in [1, n], \quad (2.13)$$

После первой реализации было представлено несколько работ по улучшению метода, а также по адаптации существующих решений для различных типов задач. В реализации [79] предлагается использовать встроенную библиотеку CURAND для генерации случайных чисел при помощи графического процессора, чтобы исключить из этой операции центральный процессор, а также снизить издержки при пересылке информации в память графического процессора. Кроме того, для ускорения работыс популяцией, информация о которой размещена в глобальной памяти, предложен вариант с использованием текстурной памяти. Текстурная память физически является частью глобальной, но при этом, так как работа с текстурами является первоочередной задачей для графических процессоров, для её обработки в графическом процессоре имеются специальные аппаратные средства ускорения.Таким образом, из модели M_1исключён CPU:

$$M_2 = (DE, GPU) \quad (2.14)$$

Кроме того, в работе исключена предварительная генерация случайных чисел, т.е. описание одной процедуры оптимизации представлено как:

$$\forall i \in [1, n], de_i = (D_i, F_i, CR_i, g_i), \quad (2.15)$$

Что упрощает итоговое преобразование:

$$f_2 = de_i \rightarrow gpu, \forall i \in [1, n], \quad (2.16)$$

В работе [80] проводится сравнительный анализ реализаций на графическом процессоре метода дифференциальной эволюции и генетического алгоритма. Для тестов используется реализация метода дифференциальной эволюции, схожая с предложенной в работе [79], но при этом в работе не используется текстурная память. Следует отметить, что в работе показано, что метод дифференциальной эволюции существенно проще, чем генетический алгоритм для реализации на графическом процессоре, прежде всего из-за наличия слабо параллельных стадий генетического алгоритма (перенос из одной популяции в другую). Кроме того, по результатам проведённых тестов, метод дифференциальной эволюции показал

более хороший результат при поиске решения для задачи планирования выполнения набора задач на кластере компьютеров, а именно позволил организовать вычисления таким образом, чтобы минимизировать общее время выполнения всех задач и максимизировать загрузку вычислительный устройств.

В работе [81] производится предварительное вычисление (в отличие от работ [79, 80]) случайных чисел с использованием графического процессора. В реализации из работы [82] графический ускоритель предлагается использовать исключительно для вычисления целевой функций. Все этапы метода дифференциальной эволюции выполняются с использованием центрального процессора. Таким образом, модель имеет вид:

$$M_3 = (DE, GPU, CPU) \quad (2.17)$$

$$CPU = (thread_1, thread_2, \dots, thread_m), \quad (2.18)$$

где$thread$ - поток центрального процессора.

Итоговое преобразование в таком случае имеет вид:

$$f_3 = (de_i \to thread_j), (g_i \to gpu), \forall i \in [1, n], \forall j \in [1, m], \quad (2.19)$$

Авторами работы [83] было отмечено, что реализация каждого этапа алгоритма при помощи отдельного вычислительного ядра является неэффективным подходом, так как основные этапы метода логически связаны и напрямую опираются на результаты предыдущих этапов. Была предложена идея объединить несколько ядер. В результате были соединены ядра I и E, также ядра M, C, R и E (далее ядро MCRE). Кроме того, авторами было отмечено, что использование фиксированного размера вычислительной CUDA сетки может приводить к значительному снижению вычислительной нагрузки на мультипроцессоры в случае, когда размер кластера векторов значительно меньше используемого размера вычислительного блока, в результате было предложено автоматическое конфигурирование вычислительной сетки в процессе выполнения. Помимо этого, была озвучена идея использования CUDA потоков для запуска ядер, которые могут быть выполнены независимо. Это касается ядер MCRE и P. Согласно изложенному подходу, для одной процедуры дифференциальной

эволюции используется два CUDA потока. Так ядра P и MCRE ставятся в очередь потока #0, при этом ядро P ставится в очередь потока #1 для вычисления матрицы с индексами векторов для выполнения мутаций для следующего запуская ядра MCRE. Таким образом, вычисления чередуются. Индексы векторов для выполнения очередного этапа мутации генерируются в процессе выполнения текущего запуска MCRE. Изменённая схема представлена на рисунке 2.2.

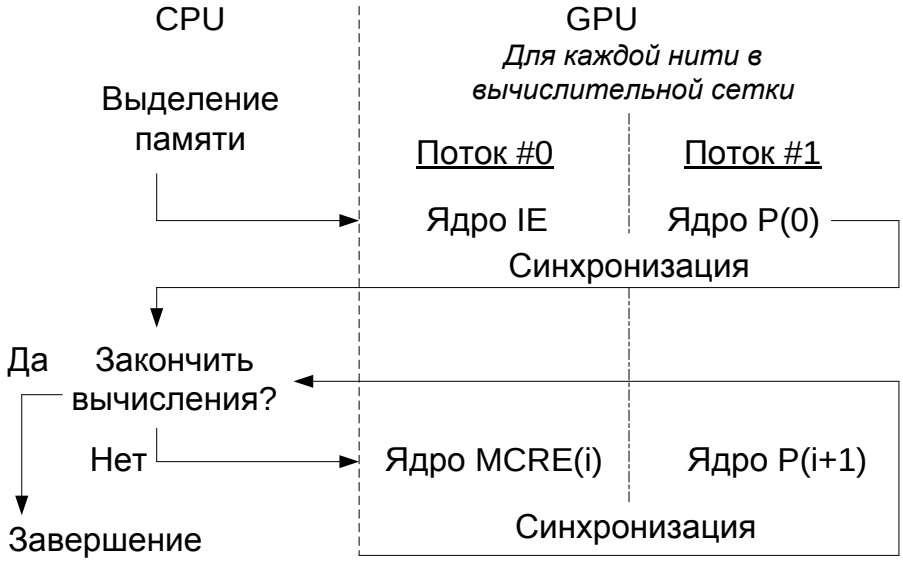


Рисунок 2.2 — Описание реализации метода дифференциальной эволюции с использованием потоков для вычисления различных ядер

С учётом использования нескольких потоков модель выглядит следующим образом:

$$M_4 = (DE, GPU) \tag{2.20}$$

Модель работы графического процессора представлена следующим образов:

$$\begin{gathered} |GPU| = 1, gpu = (blocks, threads, STREAMS) \\ blocks - var, threads - var, |STREAMS| = 2 \end{gathered} \tag{2.21}$$

Итоговое преобразование определяется отображением:

$$f_4 = \left(de_i \to gpu\right) \to stream_j, \forall i \in [1, n], \forall j \in [1,2] \tag{2.22}$$

2.2 Предлагаемая модель выполнения метода дифференциальной эволюции с использованием графических процессоров

Следует отметить ряд недостатков в существующих реализациях. Прежде всего, это наличие большого количества вызовов вычислительных ядер со стороны центрального процессора. Это налагает дополнительные издержки, что было отмечено в работе [83]. Разделение алгоритма на несколько ядер является следствием другого недостатка, а именно декомпозиции задачи с уклоном в использование большого количества вычислительных блоков. CUDA не имеет явных средств для выполнения синхронизации вычислений между различными блоками. Такая синхронизация может быть выполнена либо при использовании атомарных операций над глобальной памятью, либо при помощи разбиения вычислений на несколько ядер и их синхронный запуск. Такой подход налагает ограничения на использования CUDA потоков и полноценной загрузки мультипроцессоров.

Существующие решения не ориентированы на выполнение большого количества процедур оптимизации и не учитывают возможности систем, включающих в себя массивы графических процессоров. Помимо этого, ряд существующих реализаций используют предварительную генерацию случайных чисел.

Одним из ключевых недостатков существующих реализацийметода дифференциальной эволюции на графических процессорах является неэффективное распределение вычислений для одной задачи оптимизации по всем доступным мультипроцессорам графического процессора.

Согласно логической модели вычисления на графическом процессоре, один вычислительный блок может содержать в себе до 1024 нитей. Согласно рекомендации авторов метода дифференциальной эволюции, размер кластера векторов должен принадлежать интервалу [5D;10D]. При использовании наиболее очевидной декомпозиции задачи «одна нить обрабатывает один вектор кластера» становится понятно, что одного вычислительного блока вполне достаточно для

решения задач оптимизации с большим количеством переменных [77]. Таким образом, предлагаемая декомпозиция задачи позволяет эффективно задействовать все мультипроцессоры при решении прикладных задач, требующих выполнения большого количества процедур оптимизации. К таким задачам относится молекулярный лиганд-белковый докинг, где требуется выполнять позиционирование миллионов лигандов и их конформеров в поле биомишени. Для управления различными процедурами оптимизации необходимо использовать CUDA потоки, которые позволяют исполнять ядра независимо друг от друга. При этом для эффективного распараллеливания вычислений на несколько CUDA потоков следует руководствоваться не только стоящей задачей, но и имеющимися вычислительными ресурсами на графическом процессоре, для того чтобы наиболее эффективно управлять вычислениями.

Важным преимуществом использования одного вычислительного блока для выполнения процедуры оптимизации является возможность применять быструю встроенную барьерную синхронизацию.

Распределение вычислений на два ядра: одно для выполнения основных процедур метода дифференциальной эволюции, второе для предварительной генерации случайных чисел – и вычисление их в отдельных CUDA потоках не оправдано, так как генерация случайных чисел для выполнения этапа мутации не является затратной процедурой в сравнении с вычислением значения целевой функции. В качестве дополнительной оптимизации реализации следует исключить ядро P из вычислительной схемы и заменить вычисления индексов для формирования мутантов на вычисления в процессе выполнения метода.

Начальная инициализация векторов кластера также не является вычислительно затратной процедурой.Поэтому следует оставить инициализацию на центральном процессоре, так как это позволит выполнять процедуру более интеллектуальным образом, и в случае необходимости ввести дополнительные эмпирические соображения, обусловленные прикладной задачей, что сложнее сделать на графическом процессоре из-за дивергентных варпов, а также затрат на передачу дополнительной информации в память графического процессора.

С учётом предлагаемых изменений из (2.10) исключено множество предварительно сгенерированных случайных чисел.Таким образом, определение процедуры оптимизации представлено следующей формулой:

$$\forall i \in [1,n], de_i = (D_i, F_i, CR_i), \tag{2.23}$$

Модель работы графического процессора представлена как:

$$\forall j \in [1,m], gpu_j = \left(MX_j, STREAMS_j\right), \tag{2.24}$$

где MX_j – множество мультипроцессоров графического процессора, $STREAMS_j$ – множество CUDAпотоков, управляющих выполнением процедур оптимизации.

Распределение процедур оптимизации для одного графического процессора, согласно предлагаемой модели, представлено как:

$$P_s \subset DE, |P_s| = \left|STREAMS_j\right|, P_1 \cup P_2 \cup \ldots \cup P_r = DE,$$
$$P_1 \cap P_2 \cap \ldots \cap P_r = \emptyset, s = \overline{1,r} \tag{2.25}$$
$$f = P_s \to STREAMS_j$$

Предлагаемая схема алгоритма реализации метода дифференциальной эволюции представлена на рисунке 2.3.

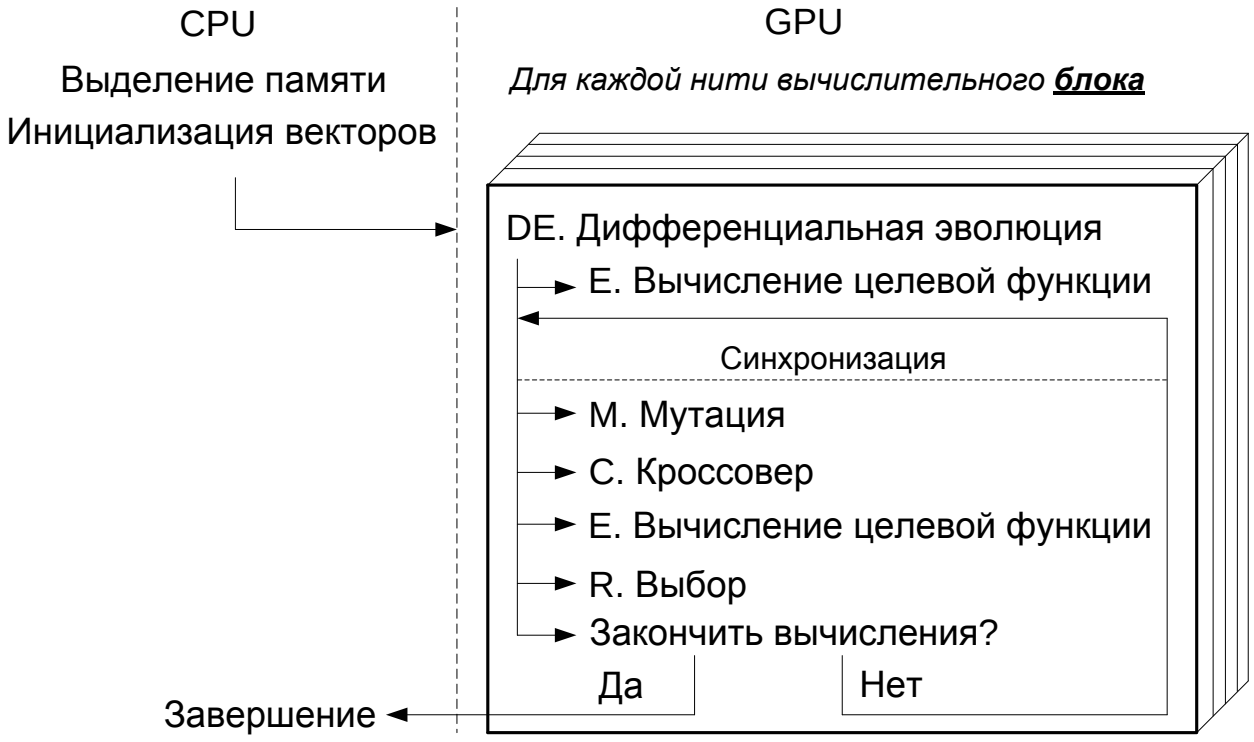


Рисунок 2.3 - Описание предлагаемой реализации метода дифференциальной эволюции с использованием одного ядра

Таким образом, предлагаемое решение в отличие от существующих позволяет реализовать выполнение процедуры оптимизации в рамках одного вычислительного блока CUDA и обеспечивает возможность выполнения нескольких процедур оптимизации одновременно на одном графическом процессоре.Это позволяет использовать встроенную быструю барьерную синхронизацию и задействовать возможности массивов графических процессоров.

2.3 Организация основных структур данных для выполнения метода дифференциальной эволюции

Для реализации метода дифференциальной эволюции поддерживаются три блока памяти для хранения информации о кластерах векторов: векторы текущей итерации, мутантные векторы, тестовые векторы. Каждый вектор текущей итерации, а также тестовый вектор имеет размерность D+1, где D количество оптимизируемых переменных. Дополнительная ячейка памяти отводится для хранения вычисленного значения функции, что не требуется для мутантных векторов. Структуры данных для кластеров векторов хранятся в глобальной памяти графического процессора. Использование разделяемой памяти может быть оправдано только при оптимизации функции, не имеющей большого количества параметров из-за ограниченного объёма. Для объединения запросов и увеличения производительности кластеры организованы в структуры данных согласно рисунку 2.4.

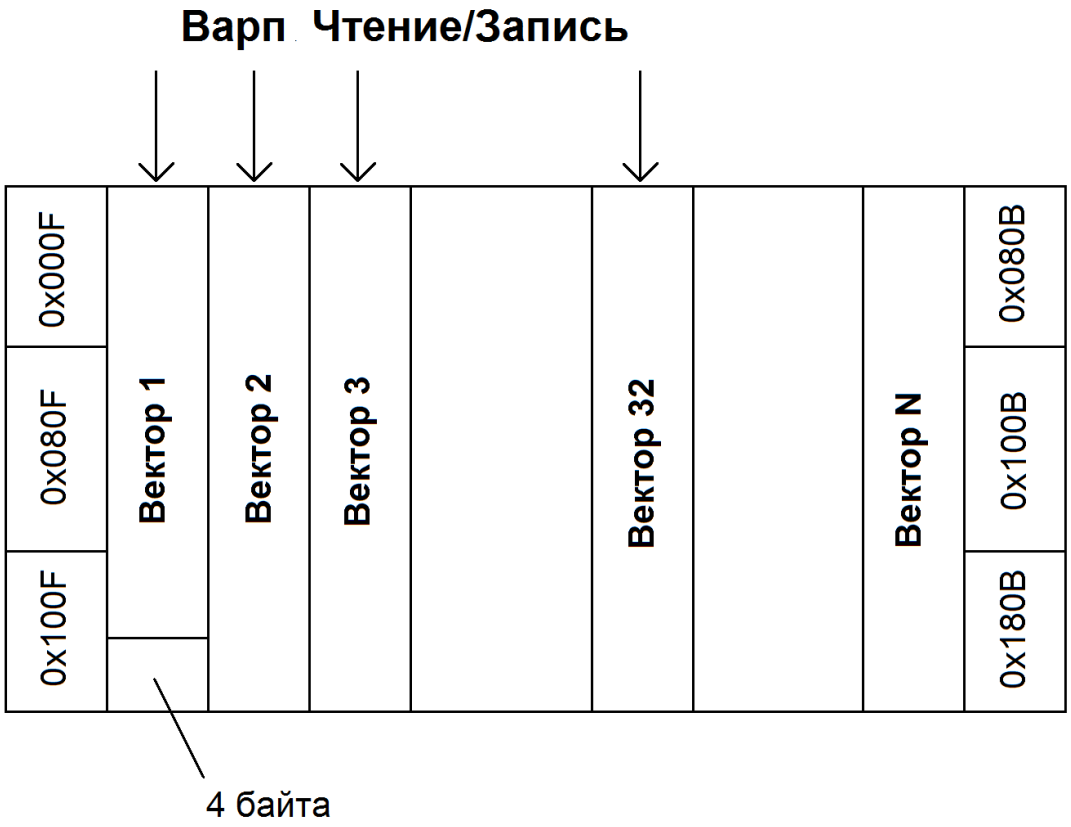


Рисунок 2.4 - Распределение кластеров векторов в памяти графического процессора для одной процедуры оптимизации

Для объединения запросов от различных нитей в пределах варпа для ячеек векторов, имеющих вещественный тип данных (*float*) происходит размещение векторов в памяти параллельно, а не последовательно друг за другом. При таком способе ячейки имеющие одинаковый индекс по всем векторам размещаются последовательно друг за другом. При таком способе организации памяти, обращения в пределах одного варпа будут происходить по выровненному адресу и к блоку размером в 128 байт, что приведёт к объединению запросов в один.

Рисунок 2.5 - Блок-схема метода дифференциальной эволюции для нити графического процессора

Во избежание гонок при параллельной работе большого количества нитей для каждой нити вычислительного блока выделяется собственный генератор случайных чисел. При этом для исключения корреляции генерируемых случайных значений до начала процедуры дифференциальной эволюции отдельное вычислительное ядро выполняет инициализация массива генераторов. Вспомогательные структуры данных, предназначенные для работы генераторов, размещаются линейным образом в глобальной памяти графического процессора так, что обращение к ним в пределах варпа является максимально выровненным. Это позволяет объединять запросы от отдельных нитей в один.

Помимо обозначенных структур, предлагаемая реализация не подразумевает других существенных затрат памяти, требующих специфического размещения.

2.4 Обобщённая схема алгоритма реализации метода дифференциальной эволюции для одного вычислительного блока

В пределах одной задачи оптимизации каждой нити одного вычислительного блока ставится в соответствие вектор приближенного решения. Все этапы метода дифференциальной эволюции для вектора выполняются только одной нитью. Для каждого вектора случайным образом (при помощи библиотеки CURAND) генерируются индексы векторов, которые будут

использованы для формирования мутантного вектора, после чего случайным образом определяется точка на векторе приближенного решения для начала выполнения кроссинговера и формирования тестового вектора. После формирования тестовыхвекторов происходит вычисление значения оптимизируемой функции. На этом этапе происходят вычисления, специфичные для решения конкретной прикладной задачи. Результаты вычислений значения оптимизируемой функции используются для формирования векторов новой итерации, после чего выполняется барьерная синхронизация всех нитей графического процессора с целью убедиться, что формирование новых приближенных решений осуществлено всеми нитями вычислительного блока. Затем выполняется обмен данными между новым приближенным решением и предыдущим приближенным решением. После обмена также осуществляетсябарьерная синхронизация всех нитей, для того чтобы начать новую итерацию метода дифференциальной эволюции с актуальной версией векторов приближенного решения. Обобщённая блок-схема алгоритма для одного вычислительного блока представлена на рисунке 2.5.

2.5 Использование CUDA потоков для организации вычислений

Предлагаемая организация выполнения метода дифференциальной эволюции позволяет обрабатывать одну процедуру оптимизации одним вычислительным CUDA блоком и, как следствие, позволяет максимально удобно и полно использовать CUDA потоки для одновременного выполнения нескольких процедур оптимизации (рисунок 2.6). Согласно спецификации CUDA, графические процессоры с уровнем вычислительных возможностей до 3.5 поддерживают до 16 CUDA потоков. Для графических процессоров с уровнем вычислительных возможностей от 3.5 доступно до 32 CUDA потоков. Руководствуясь типовым количеством мультипроцессоров на графических процессорах, ориентированных на научные вычисления, следует генерировать по одному CUDA потоку на один имеющийся мультипроцессор. Компоновка метода

дифференциальной эволюции в форме единого ядра позволяет исключить дополнительное управление ходом вычислений со стороны центрального процессора. В случае разделения процедуры оптимизации на несколько независимых вычислительных ядер (как это предложено в предыдущих реализациях) с целью синхронизации между этапами вычислений потребовалось бы либо выделение дополнительного пула потоков центрального процессора для независимого управления несколькими процедурами оптимизации, либо внесение искусственных зависимостей в процесс вычислений между различными процедурами оптимизации. При решении большого пула неоднородных задач оптимизации внесение подобных зависимостей существенно снизит производительность за счёт простоя на каждом этапе вычислений менее нагруженных процедур в ожидании завершения вычислений процедурами, оптимизирующими более сложные функции. Таким образом, требуется только один поток центрального процессора для управления вычислениями на графическом процессоре, в задачи которого входит генерация CUDA потоков, распределение нагрузки между ними и ожидание завершения вычислений в потоках. Подобное распределение вычислений позволяет повысить эффективность процесса за счёт более гибкого и независимого манипулирования процедурами нескольких оптимизаций.

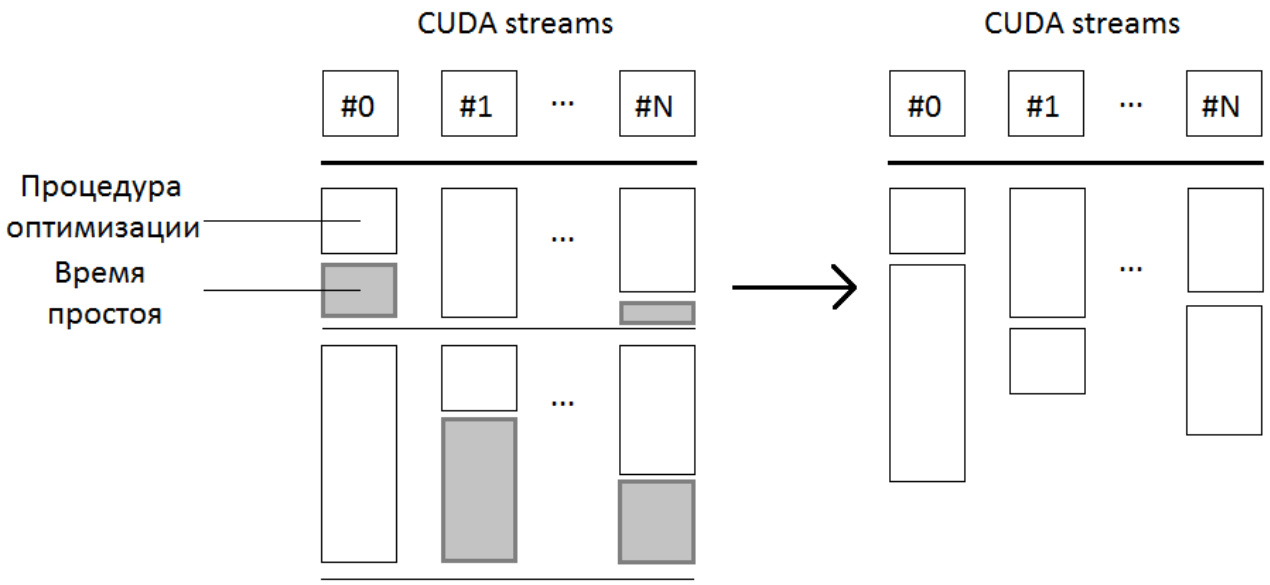


Рисунок 2.6 - Использование CUDA потоков для управления выполнением нескольких процедур оптимизации

Подобная схема также не вносит никаких дополнительных зависимостей для случая наличия нескольких графических процессоров в составе компьютера. В подобной ситуации для каждого графического процессора генерируется отдельный поток центрального процессора.

2.6 Выводы по второй главе

Предложен алгоритмреализации метода дифференциальной эволюции с использованием графического процессора, позволяющий задействовать возможности только одного мультипроцессора графического процессора, при этом полностью соответствующий требованиям метода на размер вектора приближенных решений для обеспечения сходимости. Это, в свою очередь, позволяет одновременно выполнять большое количество процедур оптимизации на одном графическом процессоре и использовать встроенное быстрое средство барьерной синхронизации в отличие от существующих подходов, в которых возможно выполнение только одной процедуры оптимизации на одном графическом процессоре.Предложенная реализация позволяет инкапсулировать

все вычисления в пределах одного вычислительного ядра и не использует случайные числа,предварительно сгенерированные на центральном процессоре, что в совокупности снижает задержи при выполнении процедуры.

Результаты, представленные в главе, опубликованы в работах [84,85].

3 Применение метода дифференциальной эволюции для выполнения молекулярного докинга с использованием графических процессоров

В главе рассмотрен сеточный подход к ускорению молекулярного докинга. Описаны достоинства и недостатки существующих подходов, а также предложена реализация, эффективно использующая ресурсы графического процессора. На основе описанного алгоритма выполнения метода дифференциальной эволюции, а также предложенного сеточного подхода разработан алгоритм выполнения молекулярного лиганд-белкового докинга с использованием графических процессоров.

3.1 Применение сеточного подхода в молекулярном докинге

Для ускорения молекулярного докинга на этапе оценки (*scoring*) используется сеточный подход [30,39-41]. Нетрудно заметить, что при выполнении докинга в процессе оптимизации расположения лиганда в сайте связывания биомишени он позиционируется различным образом, и это позиционирование так или иначе сгруппировано в пределах некоторой ограниченной области, т.е. одни и те же атомы лиганада попадают в сравнимо одинаковые точки пространства достаточно часто. Эта идея лежит в основе сеточного подхода к ускорению лиганд-белкового докинга.

Перед выполнением докинга происходит расчёт набора трёхмерных сеток силовых полей, значения в которых отражают значения компонент энергии связывания. Биомишень (или её основная часть в случае учёта подвижности цепей мишени) полагается неподвижной и размещается в некоторой сетке (как правило, кубоиде). Для сетки задаются размеры по координатным осям, шаг сетки, а также местоположение относительно биомишени. Следует отметить, что сетки вычисляют только для межмолекулярного взаимодействия, а именно для Ван-дер-Ваальсова и электростатического взаимодействия.

При вычислении набора сеток для Ван-дер-Ваальсова взаимодействия для лиганда выбираются все типы атомов, после чего для каждого типа происходит последовательное размещение атома пробы этого типа в каждой ячейки сетки. Далее происходит расчет энергии Ван-дер-Ваальса для взаимодействия биомишени и атома пробы в данной точке пространства. Данный процесс итеративно повторяется для всех типов атомов, присутствующих в лиганде.

По аналогии выполняется вычисление энергии электростатического взаимодействия. В отличие от Ван-дер-Ваальсова взаимодействия в данном случае требуется вычисление только одной сетки, так как заряд конкретного атома лиганда можно исключить на этапе расчёта сеток.

После выполнения процедуры имеется некоторый набор сеток, который поступает на вход алгоритма докинга. В результате при позиционировании лиганда в активном сайте белка нет необходимости каждый раз пересчитывать для каждого атома лиганда его энергию взаимодействия с атомами биомишени. Достаточно взять соответствующие предвычисленные значения из сеток.

3.2 Существующие реализации сеточного подхода

В некоторых случаях вычисление сеток силовых полей является частью программы, что не позволяет использовать ранее предвычисленные сетки для различных лигандов. В связи с этим целесообразно использовать отдельное программное решение для вычисления сеток. Такую возможность предоставляет программа AutoGrid, которая выполняет вычисление сеток силовых полей (электростатики и взаимодействия Ван-дер-Вааласа) для программы докинга AutoDock [30]. Существенным недостатком является ориентированность программы на одноядерную архитектуру.Помимо этого, AutoDock не позволяет пользователям самостоятельно устанавливать параметры для используемых типов атомов.

Предпринимались попытки перенести вычисления сеток силовых полей на графические процессоры. Одна из первых работ по применению графических

процессоров представлена в [69]. Предложенное решение не столько ориентировано на вычисление сеток силовых полей, сколько предоставляет теоретическую базу для вычисления парных потенциалов электростатического взаимодействия. В работе предложены два варианта декомпозиции: наивная, с использованием мультиблочной вычислительной сетки; с использованием разделяемой памяти. В первом случае отдельная CUDA нить вычисляет отдельный потенциал. При втором способе декомпозицииметод вычислений сохраняется, но при этом информация об атомах лиганда копируется в разделяемую память, что позволяет уменьшить время работы с памятью за счёт сокращения обращения к глобальной памяти.

Более полноценное решение представлено в работе [70]. Авторы демонстрируют решение для вычисления сеток электростатического и Ван-дер-Ваальсова взаимодействия с использованием графических процессоров. В работе не приводится информация о применяемой декомпозиции задачи вычисления отдельной сетки. При этом авторами предложено разделение вычислений сеток для разных компонент энергии между несколькими графическими процессорами. В работе заявлено увеличение производительности относительно однопроцессорного решения для вычисления сетки электростатического взаимодействия от 33 до 287 раз (для различный компьютеров и различных пар биомишень/лиганд),а также увеличение производительности для вычисления сеток Ван-дер-Ваальсова взаимодействия от 47 до 193 раз. При этом ускорение от использования нескольких графических процессоров составило 1,4. К явным недостаткам решения следует отнести некорректную декомпозицию задачи при использовании нескольких графических процессоров, что подтверждается незначительным приростом производительности. Это можно объяснить более сложным процессом вычислений энергии Ван-дер-Ваальсова взаимодействия, кроме того, в отличие от электростатического взаимодействия в данном случае требуется вычислять несколько сеток для каждого используемого типа атомов. В результате предложенная схема декомпозиции не обеспечивает правильную балансировку нагрузки между доступными графическими процессорами.

3.3 Алгоритм расчёта силовых полей межмолекулярного взаимодействия с помощью сеток на графических процессорах

Использование предварительно вычисленных сеток силовых полей для биомишени позволяет существенным образом увеличить скорость выполнения докинга за счёт сокращения операций вычисления энергий взаимодействия молекул в конкретной конформации. При разработке нового решения для вычисления сеток силовых полей, использующего возможности графических процессоров, были сформулированы следующие требования, которым оно должно удовлетворять:

- решение следует изолировать от используемого силового поля, которое должно устанавливаться во входных параметрах программы для обеспечения универсальности за счёт простой смены или корректирования силового поля, что позволит настраивать программное обеспечение в зависимости от требуемых химических параметров среды моделирования;
- программа должна быть рассчитана на вычисление сеток силовых полей для большого набора биомишений;
- система балансировки нагрузки должна быть устроена таким образом, чтобы прозрачно для пользователя равномерно распределять требуемые вычисления по всем доступным и подходящим для вычислений ресурсам.

Задачу можно сформулировать следующим образом. На входе процедуры расчёта сеток имеется множество биомишений *TARGETS*, где каждому элементу targets_i соответствует множество лигандов $LIGANDS_i$, где каждому лиганду ligand_{ij} соответствует такое множество типов атомов $ATOM\_TYPES_{ij}$, что каждый тип atom_type_{ijk} хотя бы один раз присутствует в лиганде. В результате вычислений необходимо получить некоторое множество сеток *GRIDS*, элемент которого grid_{ijk} является результатов вычисления сетки силового поля для атома типа *k*, принадлежащего лиганду *j* и взаимодействующего с белком *i*. Сетки силового поля для одного типа атомов, присутствующего в различных лигандах и

взаимодействующих с одной и той же биомишенью, являются эквивалентными, так как имеют одинаковые Ван-дер-Ваальсовы параметры,поэтому справедливо:

$$\text{atom\_types}_{ijk} = \text{atom\_types}_{lmn} \rightarrow \text{grid}_{ijk} = \text{grid}_{lmn}. \quad (3.1)$$

Таким образом, можно уменьшить количество вычислений, заменив парное вычисление сеток для каждой биомишени targets_i из множества *TARGETS* с соответствующими ей лигандами из множества $LIGANDS_i$ на вычисления сеток для биомишени i и типов атомов, присутствующих хотя бы в одном лиганде из множества $LIGANDS_i$.

Отдельно стоит рассмотреть вычисление сеток для электростатического взаимодействия. С целью дополнительного упрощения можно вынести заряд атома лиганда из процесса вычислений. В результате необходимо будет вычислить только одну сетку электростатического взаимодействия для биомишени. В результате для каждой биомишени targets_i необходимо вычислить

$$\left|ATOM\_TYPES_{i0} \cup ATOM\_TYPES_{i1} \cup \ldots \cup ATOM\_TYPES_{i(N-1)}\right| + 1, N \coloneqq |LIGANDS_i| \quad (3.2)$$

сеток.

Так как энергия взаимодействия в некоторой точке пространства не зависит от значения энергии в окружающих её точках как для Ван-дер-Ваальсова взаимодействия, так и для электростатического взаимодействия, процесс вычислений отдельной сетки имеет высокую степень параллелизма по данным. Фактически значение энергии взаимодействия для каждой точки сетки вычисляется независимо. Кроме того, вычисления отдельных сеток даже для одинаковой биомишени также полностью независимы.

К решаемой задаче была применена двойная декомпозиция: на уровне сеток и на уровне ячеек сеток. Первый уровень декомпозиции позволяет эффективно балансировать нагрузку при использовании нескольких графических процессоров. Декомпозиция на уровне сеток схематически показана на рисунке 3.1.

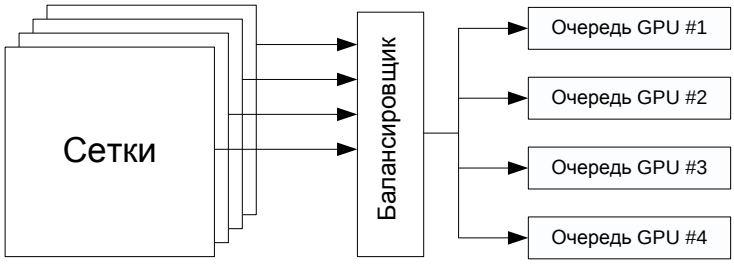


Рисунок 3.1 - Декомпозиция на уровне вычисления отдельных сеток

Разбиение сеток сделано относительно определённой биомишени таким образом, чтобы вычисление сеток для одной биомишени не выполнялось на разных графических процессорах. Это необходимо с целью сокращения накладных расходов на выделение глобальной памяти графических процессоров и пересылки информации о биомишени. При разбиении вычислений сеток для одной биомишени между различными графическими процессорами выделять память, а также переносить информацию о трёхмерной структуре, типах атомов, их зарядах и Ван-дер-Ваальсовых параметрах необходимо для каждого графического процессора, что является неоправданно затратной процедурой. Таким образом, в очередь GPU выставляется группа вычислений электростатической сетки и сеток Ван-дер-Ваальсова взаимодействия для одной биомишени.

Второй уровень декомпозиции позволяет эффективно использовать доступные ресурсы отдельного графического процессора и настраивать вычисления таким образом, чтобы полностью задействовать доступные мультипроцессоры. Вычислительная сложность процедуры расчёта одной сетки для одной биомишени равна $O(n*m)$, где n— количество ячеек в сетке, а m— это количество атомов в биомишени. Количество ячеек может достаточно произвольно меняться в зависимости от исследуемой области биомишени и требуемой точности, но, как правило, находится в диапазоне от 64000 (сетка размерностью 40*40*40) до 1000000 (сетка размерностью 100*100*100). Так как вычислительная нагрузка велика, алгоритм является масштабируемым, вследствие чего рост количества отдельных вычислителей не приведёт к их простою, а из-за специфики диспетчеризации отдельных нитей на графических

процессорах увеличение параллельно работающих вычислителей не даст существенного роста накладных расходов на запуск вычислений. Кроме того, повышению масштабируемости алгоритма способствует отсутствие необходимости выполнять какую-либо синхронизацию вычислений отдельной сетки в процессе работы. Исходя из этого, при организации вычислительного процесса в предлагаемом решении используется мультиблочная сетка на графическом процессоре. Количество вычислительных CUDA блоков при этом выбирается исходя из количества доступных мультипроцессоров и загруженность (*occupancy*) для каждого вычислительного ядра.

Для вычисления сеток для одной биомишени достаточно подготовить два CUDA ядра. Первое предназначено для вычисления сетки для электростатического взаимодействия, второе предназначено для вычисления сеток Ван-дер-Ваальсова взаимодействия. Псевдокод для вычисления сетки электростатического взаимодействия одной CUDA нитью представлен на рисунке 3.2.

```
cell_index:=thread_index+block_size*block_index
while cell_index<grid_cell_count
        (x,y,z):=get3DCoordinate(cell_index);
        (biotarget_atom_index,energy):=0;
        while biotarget_atom_index< biotarget_atoms_count
                (atom_x,atom_y,atom_z):=getBiotargetAtomCoordinate(biotarget_atom_index);
                atom_charge:=getBiotargetAtomCharge(biotarget_atom_index);
                energy:= energy+calculatePairwasePotential(x,y,z,atom_x,atom_y,atom_z,atom_charge);
                biotarget_atom_index:=biotarget_atom_index+1;
        endwhile
        cell_index:=cell_index+blocks_count*block_size;
endwhile
```

Рисунок 3.2 - Псевдокод вычисления сетки электростатического взаимодействия

Так как порядок вычисления ячеек сетки взаимодействия не имеет значения, фактическое представление сетки как трёхмерного объекта не играет существенной роли. В связи с этим каждая сетка представлена в памяти как одномерный массив размерностью *grid_x_size*grid_y_size*grid_z_size*. Как видно из рисунка 3.3, каждая нить инициализирует свои вычисления в зависимости от

принадлежности к блоку и положению внутри блока.Это обеспечивает пространственную локальность обращения к глобальной памяти графического процессора при накоплении информации об энергии. В результате все запросы к глобальной памяти являются объединёнными (рисунок 3.3, где b – номер вычислительного CUDA блока в пределах CUDA сетки, t – номер CUDA нити в пределах блока).

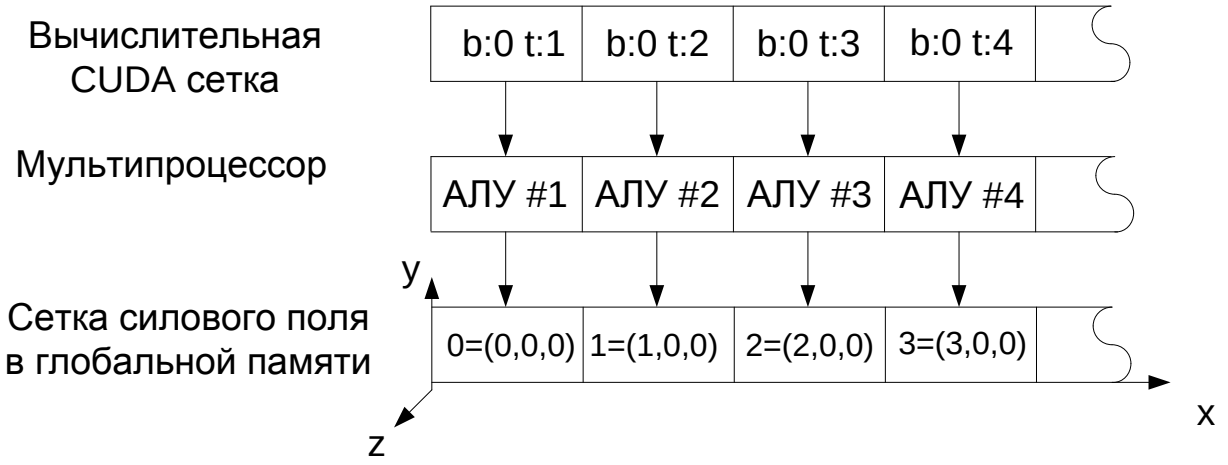


Рисунок 3.3 - Декомпозиция вычисления отдельной сетки силового поля и её представление в глобальной памяти графического процессора

Перевод линейной координаты ячейки сетки (*cell_index*) в её реальные трёхмерные координаты (*x*,*y*,*z*) осуществляется согласно:

$$\begin{gathered} x = (cell\_index \,\%\, grid\_size) * grid\_step \; + grid\_x\_left \\ y = ((cell\_index \,\%(grid\_x\_size * grid\_y\_size))/grid\_x\_size) * grid\_step \; + grid\_y\_left \\ z = (cell\_index/(grid\_x\_size * grid\_y\_size)) * grid\_step + grid\_z\_left, \end{gathered} \tag{3.3}$$

где *grid_step* – шаг сетки, (*grid_x_left*, *grid_y_left*, *grid_z_left*) – координаты левого нижнего угла сетки в ангстремах.

Псевдокод для вычисления сеток силовых полей для Ван-дер-Ваальсова взаимодействия представлен на рисунке 3.4. В данном случае добавляется дополнительный цикл для прохода по всем типам атомов (пробам), которые были получены из силового поля. При этом следует отметить, что дополнительно для выполнения вычислений в ядро передаются Ван-дер-Ваальсовы параметры всех типов атомов (atom_vdw_radius – Ван-дер-Ваальсов радиус, atom_vdw_epsilon – размерность энергии (*well-depth*)).

Так как на практике различные типы атомов силового поля, имеющие одинаковые Ван-дер-Ваальсовы параметры, ничем не отличаются при выполнении процедур докинга, для них выполняется расчет только лишь одной сетки Ван-дер-Ваальсова взаимодействия в целях экономии времени вычисления и используемой памяти. При этом для дальнейшего использования формируется специальная структура, отображающая все типы атомов на реальные вычисленные сетки силовых полей. Механизм отображения схематически представлен на рисунке 3.5.

```
cell_index:=thread_index+block_size*block_index
while cell_index<grid_cell_count
        (x,y,z):=get3DCoordinate(cell_index);
        biotarget_atom_index:=0;
        while biotarget_atom_index< biotarget_atoms_count
                (atom_x,atom_y,atom_z):=getBiotargetAtomCoordinate(biotarget_atom_index);
                (atom_vdw_radius,atom_vdw_epsilon):=getBiotargetAtomVdw(biotarget_atom_index);
                (probe_index,energy):=0
                while probe_index<probes_count
                        energy:= energy+
                                calculatePairwasePotential(x,y,z,
                                        atom_x,atom_y,atom_z,
                                        atom_vdw_radius,atom_vdw_epsilon);
                        probe_index:=probe_index+1;
                endwhile
                biotarget_atom_index:=biotarget_atom_index+1;
        endwhile
        cell_index:=cell_index+blocks_count*block_size;
endwhile
```

Рисунок 3.4 - Псевдокод вычисления сеток Ван-дер-Ваальсова взаимодействия.

Для вычисления одной сетки требуется объём памяти, равный количеству ячеек в сетки на соответствующий тип данных. Для сеток достаточно использовать вычисления с одинарной точностью.

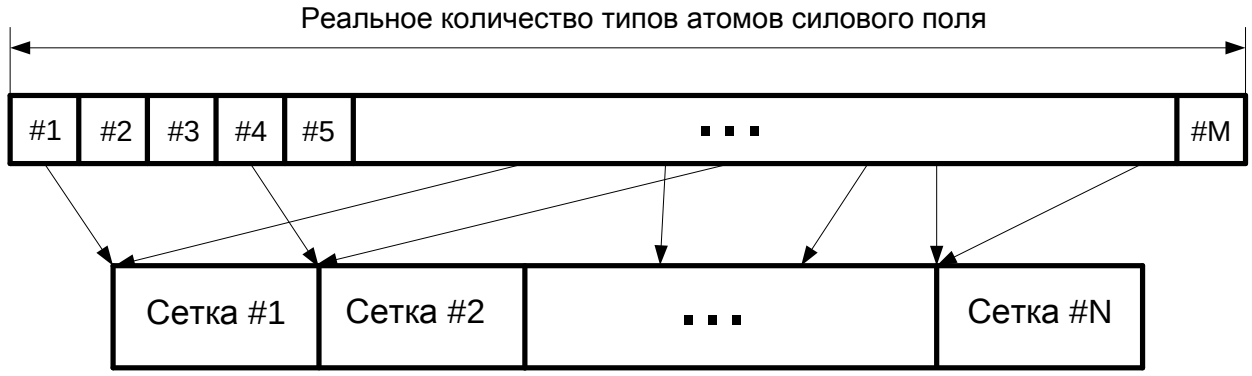


Рисунок 3.5 - Механизм отображения типов атомов силового поля на реально вычисленные сетки силовых полей Ван-дер-Ваальсова взаимодействия

Таким образом, в соответствии с указанными выше размерами сеток занимаемый объём не превышает 4 МБ, что позволяет разместить в памяти графического процессора все сетки силовых полей для нескольких биомишений. Подавляющее большинство современных графических процессоров, ориентированных на научные расчёты, а также графические процессоры предыдущего поколения имеют не менее 2ГБ глобальной памяти. Такого объёма достаточно для вычисления сеток силовых полей с размерностью по всем измерениям до 256.

Для полной загрузки каждого мультипроцессора графического процессора (обеспечение высоко уровня его загруженности) необходимо обеспечивать для него достаточный объём заданий, подготовленных к вычислению, т.е. размещённых надлежащим образом в памяти графического процессора. В данном случае ограничением является получение информации о биомишенях и вывод вычисленных сеток силовых полей на диск. Для преодоления данной проблемы и сглаживания задержек, обусловленных вводом/выводом, предлагается специальный менеджер времени выполнения (*runtime*). Менеджер представляет собой разделяемый ресурс, который используется тремя потоками центрального процессора:

– i_thread –ввод информации с диска и подготовка для вычислений;
– d_thread – управление вычислениями на графическом процессоре;
– o_thread – вывод результатов на диск.

Менеджерподдерживает две очереди, а также методы для работы с ними:

– i_queue– очередь для задач, готовых выполняться на графическом процессоре;

– o_queue – очередь для задач, которые были посчитаны на графическом процессоре.

Процесс выглядит следующим образом. i_thread получает информацию о биомишенях и сетках с диска. Производит типизацию биомишени согласно используемому силовому полу, после чего переупаковывает биомишень в формат, подходящий для вычислений на графическом процессоре. Координаты атомов биомишени, их заряд, а также типы согласно силовому полу переупаковываются в одномерный массив. Помимо этого, подготавливается представление параметров сетки для использования на графическом процессоре (независимо от биомишени) в случае, если такой сетки не было ранее. После чего i_thread формирует задачи на вычисления сеток электростатического и Ван-дер-Ваальсова взаимодействия и помещает их в очередь i_queue. Менеджер вычислений содержит в себе информацию о всех имеющихся и запущенных на вычисления задачах, а также об объёме используемой памяти графического процессора. Базируясь на этой информации, поток i_thread может либо разделять вычисления сеток (например, разделять вычисления сеток Ван-дер-Ваальсова взаимодействия для нескольких типов), либо полностью останавливать процесс постановки задач до момента освобождения требуемого объёма памяти.

Каждое представление биомишени и параметров сетки имеет собственные счётчики ссылок, которые увеличиваются при постановке задач и уменьшаются при их завершении. В обязанности процесса i_thread входит мониторинг счётчиков ссылок. В случае необходимости им выполняется освобождение памяти из-под неиспользуемых биомишеней и сеток. Преждевременное удаление информации о биомишенях или параметров сеток исключается, так как нагрузка распределяется на этапе балансировки. Таким образом, на этапе выполнения нагрузка сгруппирована по биомишеням и требуемым сеткам. Поток d_thread выполняет все действия по управлению вычислениями на графическом процессоре: получает задачи из очереди i_queue, загружает информацию в память

графического процессора, инициирует вычисления, выгружает завершённую задачу в память центрального процессора. Поток o_thread получает завершённые задачи из очереди o_queue и выгружает информацию на диск, после чего освобождает память, занимаемую результатами вычислений, и уменьшает соответствующие счётчики ссылок. Схематически менеджер времени выполнения представлен на рисунке 3.6.

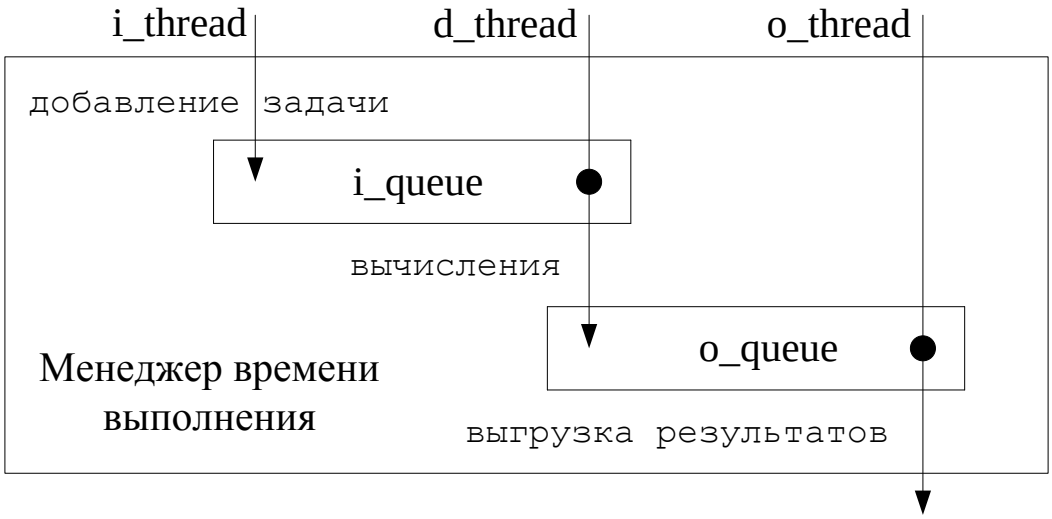


Рисунок 3.6 - Схема менеджера времени выполнения

Если компьютер имеет несколько графических процессоров, для каждого создаётся собственный менеджер времени выполнения.

3.4 Алгоритм выполнения молекулярного лиганд-белкового докинга с использованием графических процессоров

Основной целью разрабатываемого решения является быстрый скрининг большого набора лигандов при изучении некоторой определённой биомишени, так как это наиболее востребованная задача при поиске кандидатов в лекарства. В связи с этим модель должна максимально использовать все доступные ресурсы графического процессора, равномерно распределять вычислительную нагрузку и использовать все доступные графические процессоры системы. Программное обеспечение должно быть ориентировано на выполнение гибкого лиганд-

белкового докинга с учётом подвижности лиганда по всем торсионным углам. При этом определение вращающихся связей в лиганде должно осуществляться автоматически. Предлагаемая модель выполнения оптимизации с использованием метода дифференциальной эволюции хорошо соответствует этим требованиям.В контексте молекулярного лиганд-белкового докинга предложенная ранее модель выполнения метода дифференциальной эволюции на графических процессорах будет выглядеть следующим образом:

$$D = (LIGANDS, TARGET, GPU), \quad (3.4)$$

$$LIGANDS = (ligand_1, ligand_2, \dots, ligand_n), \quad (3.5)$$

$$GPU = (gpu_1, gpu_2, \dots, gpu_m), \quad (3.6)$$

где $LIGANDS$– множество всех лигандов, которые необходимо обработать, т.е. провести процедуру оптимизации; GPU –множество доступных графических процессоров. При этом:

$$\forall i \in [1, n], ligand_i = (F_i, CR_i, ATOMS_i, TORSIONS_i, clash_i), \quad (3.7)$$

где F_i – константа мутации ($F_i \in [0,2]$) , CR_i –константа кроссинговера для i-ой процедуры оптимизации ($CR_i \in [0,1]$), $ATOMS_i$–множество всех атомов лиганда, $TORSIONS_i$ –множество всех торсионных углов лиганда, $clash_i$ - карта складывания лиганда.

Размер популяции определяется как:

$$|V_i| = [5(6 + |TORSIONS_i|), 10(6 + |TORSIONS_i|)] \quad (3.8)$$

Модель работы графического процессора представлена следующим образов:

$$\forall j \in [1, m], gpu_j = \left(MX_j, STREAMS_j, GRIDS_j\right), \quad (3.9)$$

где MX_j – множество мультипроцессоров графического процессора; $STREAMS_j$ – множество CUDAпотоков, управляющих выполнением процедур оптимизации; $GRIDS_j$ – множество сеток силовых полей для биомишени, представленных на графическом процессоре.

Распределение вычислений для одного графического процессора, согласно предлагаемой модели, представлено как:

$$R_s \subset LIGANDS, |R_s| = |STREAMS|, R_1 \cup R_2 \cup \ldots \cup R_r = DE,$$
$$R_1 \cap R_2 \cap \ldots \cap R_r = \emptyset, s = \overline{1,r} \quad (3.10)$$
$$f = R_s \rightarrow STREAMS_j$$

Необходимо оптимизировать метод дифференциальной эволюции под задачу докинга. При выполнении лиганд-белкового докинга в качестве оптимизируемых параметров будут выступать 6 стандартных степеней свободы (смещение всего лиганда по координатным осям и его вращение вокруг координатных осей), а также в случае необходимости по одной переменной на каждый торсионный угол (рисунок 3.7). Предложенная параллельная декомпозиция метода дифференциальной эволюции позволяет обрабатывать один лиганд одним вычислительным CUDA блоком и, как следствие, позволяет максимально удобно и полно использовать CUDA потоки для одновременной обработки нескольких лигандов.

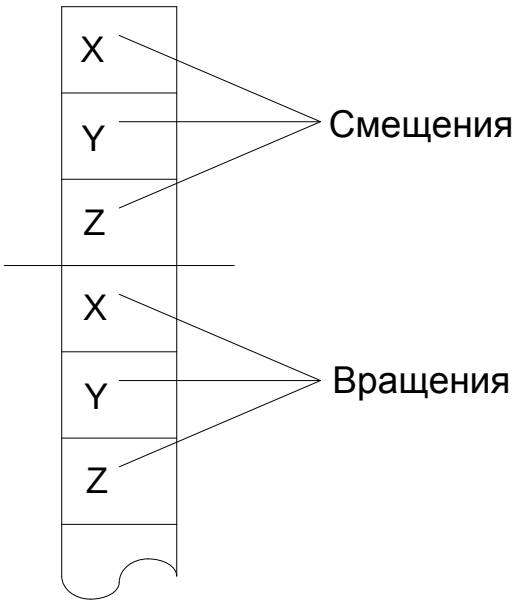


Рисунок 3.7 - Вектор приближенного решения

Сугубо специфичным для лиганд-белкового докинга является этап вычисления целевой функции, который разделён на две стадии. Прежде всего каждой нитью графического процессора выполняется трансформация лиганда согласно вектору приближенного решения, после чего выполняется собственно вычисление целевой функции. Трансформация лиганда включает в себя несколько этапов. Сначала в случае наличия выполняется поворот по всем торсионным углам. После этого выполняется поворот всего лиганда вокруг координатных осей. Затем выполняется смещение всего лиганда по координатным осям. Так как при выполнении итераций метода дифференциальной эволюции значения переменных приближенного решения меняются произвольным образом, трансформация лиганда может привести к формированию нефизической трёхмерной структуры при изменении некоторого торсионного угла таким образом, что часть лиганда будет повёрнута «в себя», либо к выходу лиганда за пределы заданной сетки, ограничивающей сайт связывания биомишени. Поэтому до вычисления целевой функции дополнительно следует провести проверку на расположение лиганда в пределах трёхмерной сетки, а также на столкновение (*clash*). В случае если структура будет признана подходящей, необходимо произвести вычисление целевой функции.

Рисунок 3.8 - Блок-схема алгоритма вычисления целевой функции

Для ускорения выполнения докинга используется сеточный подход. Так как сетки силового поля вычисляются с

некоторым детерминированным шагом, положения атомов лиганда после трансформации не может полностью соответствовать ячейкам сетки. Таким образом, для вычисления значения энергии атомов необходимо применять интерполяцию, в данном случае трилинейную интерполяцию.

Вычисление значения целевой функции производится на каждой итерации метода дифференциальной эволюции, а также при запуске метода для вычисления целевой функции для инициализированных значений векторов. Схема алгоритма вычисления значения целевой функции для отдельной CUDA нити графического процессора представлена на рисунке 3.8.

Используемая математическая модель лиганд-белкового докинга имеет следующую форму:

$$E = E_{elec} + E_{vdw} = 18{,}22 \sum_{i<j} \frac{q_{target\ ,i} q_j}{r_{ij}} + \sum_{i<j} \varepsilon_{ij} \left[\left(\frac{r_{0ij}}{r_{ij}} \right)^{12} - 2 \left(\frac{r_{0ij}}{r_{ij}} \right)^{6} \right] \quad (3.11)$$

E_{elec} – компонента, моделирующая электростатическое взаимодействие; E_{vdw} – компонента, моделирующая Ван-дер-Ваальсово взаимодействие; $q_{target\ ,i}$ - заряд iго атома биомишени, умноженный на 18,22; q_j - заряд атома лиганда; r_{ij} - расстояние между атомами в ангстремах; ε_{ij} - размерность энергии для пары атомовiи j; r_{0ij} - эталонное Ван-Дер-Ваальсово расстояние между парами атомов iи j. Размерность энергии для пары атомов вычисляется как квадратный корень из произведения размерностей энергии для отдельных атомов. Эталонное расстояние для пары атомов вычисляется как сумма эталонных расстояний для отдельных атомов.

Следует отметить, заряды атомов аминокислотных остатков представляются в силовом поле умноженными на 18,22 (что соответствует подходу, предложенному в AMBER[62, 71, 72]), в связи с чем для вычисления энергии в ккал/моль переводной коэффициент 332 заменён на 18,22.

3.5 Особенности программной реализации лиганд-белкового докинга с использованием графических процессоров

3.5.1 Организация основных структур данных

В качестве входного формата для представления биомишени выбран формат PDB, так как это наиболее популярный и открытый формат представления трёхмерных структур молекул. Существует значительное количество экспериментально полученных структур молекул, представленных в этом формате. В качестве входного формата для представления лиганда выбран формат mol2, так как он также открыт и является одним из двух самых распространённых форматов представления лигандов. В качестве силового поля также, как и при реализации вычисления сеток силовых полей, используется силовое поле AMBER.

Разрабатываемое решение использует сеточный подход для ускорения докинга, предложенный и реализованный в разделе 3.1. Используются структуры данных для организации сеток силовых полей, а также механизм отображения типов атомов на реально вычисленные сетки Ван-дер-Ваальсова взаимодействия.

Так как, согласно реализации, в каждый момент времени происходит обращение всех нитей вычислительного блока к одной и той же ячейке памяти, содержащей информацию о биомишени или лиганде (трёхмерные координаты, заряд, тип атома), то эта информация представляется в виде двух одномерных массивов для биомишени и каждого лиганда. Первый одномерный массив состоит из ячеек типа float и содержит в себе последовательно информацию о координатах X, Y, Z всех атомов молекулы, а также их заряды. Второй одномерный массив содержит в себе информацию о типах атомов. Типы атомов представлены как целые числа размерностью в один байт, чего вполне достаточно для размещения информации обо всех типах атомов рассмотренных силовых полей.

На каждой итерации метода дифференциальной эволюции выполняется преобразование исходной трёхмерной структуры лиганда согласно вектору приближенного решения. В связи с этим каждая нить вычислительного блока

должна поддерживать структуру данных для хранения рансформированного лиганда. На основании этой структуры будет выполняться расчет энергии взаимодействия. Организация памяти под координаты трансформированного лиганда совпадает с организацией памяти для векторов метода дифференциальной эволюции аналогично с целью объединения запросов к глобальной памяти. Вместо векторов последовательно размещаются трёхмерные координаты всех атомов лиганда.

Более специфичной организации памяти требуют торсионные углы лиганда. Для выполнения вращения по торсионному углу необходимо наличие информации о паре центральных атомов составляющих торсионных углов, так как именно их координаты для конкретного положения лиганда будут определять ось вращения по торсионному углу. Кроме того, требуется информация об атомах лиганда, которые будут подвержены вращению по торсионному углу. Данная информация не является однородной для набора лигандов, более того, информация для торсионных углов в пределах одного лиганда также не является однородной из-за различного количества атомов лиганда, которые необходимо вращать. При этом следует отметить, что при трансформации лиганда, а именно при вращении торсионных углов, не имеет значения (в рамках процедуры оптимизации) порядок вращения углов. Кроме того, каждый атом лиганда трансформируется независимо. В результате процесс вращения по торсионным углам и, как следствие, структура данных для представления торсионных угловмогут быть организованы последовательно (рисунок 3.9).

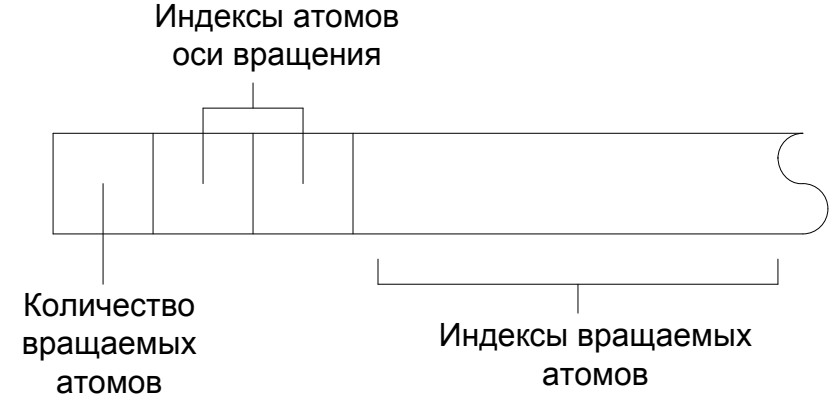


Рисунок 3.9 - Организация памяти для хранения информации о торсионных углах лиганда

Все торсионные углы хранятся последовательно в одном одномерном массиве и представляют собой набор из трёх основных компонент: количество вращаемых атомов для угла; индексы атомов, на основании которых будет определяться ось вращения торсионного угла; индексы атомов, которые необходимо поворачивать по торсионному углу. Такая организация памяти позволяет при трансформации лиганда по торсионным углам, во-первых, выполнять обращения всех CUDA нитей к одной и той же ячейке памяти, представляющей торсионный угол, и, во-вторых, в один момент времени трансформировать один и тот же атом лиганда, информация о котором для всех CUDA нитей вычислительного блока, как указывалось ранее, хранится в выровненной области памяти. Подобная организация памяти для торсионных углов также позволяет объединять все запросы к глобальной памяти графического процессора, что ускоряет обращение к глобальной памяти в пределах одного варпа до 32 раз.

3.5.2 Ограничения по используемым графическим процессорам

Для выполнения вычислений может быть использован практически любой графический процессор с уровнем вычислительных возможностей от 2.0 и выше, чему соответствуют все современные графические процессоры, а также графические процессоры предыдущих поколений. При этом следует понимать, что скорость решения поставленной задачи в рамках предложенной реализации напрямую зависит от количества доступных мультипроцессоров благодаря декомпозиции вычислений на уровне набора лигандов «один лиганд – один вычислительный блок – один CUDA поток».Основные требования предъявляются по объёму глобальной памяти графического процессора прежде всего для размещения сеток силовых полей. По умолчанию используется силовое поле, требующее вычисления 34 сеток Ван-дер-Ваальсова взаимодействия и одной сетки для электростатического взаимодействия. При использовании сеток

размерностью 128 по всем измерениям их размещение в глобальной памяти потребует 280 Мбайт. Помимо сеток силовых полей, для обработки одного лиганда необходимо три кластера векторов. Так как процедура организована на уровне одного вычислительного блока графического процессора, кластер может содержать в себе до 1024 векторов, что соответствует максимальному количеству CUDA нитей для одного вычислительного блока. Согласно рекомендациям, на выбор параметров для метода дифференциальной эволюции максимальное количество оптимизируемых параметров, т.е. размер одного вектора, равно 100 [77].Таким образом, максимальный объём глобальной памяти, требуемой для размещения кластеров векторов, не превышает 5 мегабайт.

Помимо этого, каждая нить графического процессора должна иметь структуру для трансформации лиганда. Как правило, типовой лиганд много меньше 150 атомов. Таким образом, для поддержания структуры для выполнения трансформации лиганда всеми нитями вычислительного блока необходимо 150 ячеек под каждую координату атома для всех 1024 нитей, т.е. менее двух мегабайт на один лиганд. Исходя из максимального количества поддерживаемых CUDA потоков, на современных графических процессорах одновременно в обработке может находиться до 32 лигандов. Помимо памяти под сетки силовых полей, графический процессор должен располагать ещё приблизительно 100 мегабайтами глобальной памяти. В результате общие требования по глобальной памяти много меньше 512 мегабайт, чему соответствуют все графические процессоры, ориентированные на научные вычисления.

Следует отметить, что предлагаемые для расчета верхние границы (за исключением аппаратного ограничения на количество нитей вычислительного блока) для количества атомов лиганда, оптимизируемых параметров и т.п. выбраны для примера как достаточно большой предел и не являются встроенными в предполагаемое решение ограничениями. Таким образом, предлагаемое решение может быть легко масштабировано на большие вычислительные ресурсы в случае расширения аппаратных ограничений.

3.5.3 Использование массива графических процессоров

На начальном этапе процесса происходит подбор подходящих графических процессоров, установленных в компьютере на основании соответствия уровню вычислительных возможностей и объёму свободной глобальной памяти. После этого на основе набора подходящих графических процессоров происходит равномерное распределение лигандов. При этом распределение лигандов выполняется исходя из количества мультипроцессоров на каждом графическом процессоре. Для каждого графического процессора определяется набор лигандов, равный двойному количеству доступных мультипроцессоров.

После распределения нагрузки для каждого графического процессора создаётся отдельный поток центрального процессора. Каждый поток получает информацию об обрабатываемой биомишени, силовом поле и выполняет вычисление всех требуемых сеток силового поля. Распределение вычисления сеток силовых полей между несколькими графическими процессорами неоправданно, так как для одной биомишени процесс вычисления сеток силовых полей является несравнимо быстрой процедурой по сравнению с обработкой набора лигандов.Помимо этого, серия пересылок больших объёмов информации между глобальной памятью всех графических процессоров и памятью центрального процессора создаст излишние накладные расходы. После вычисления сеток силовых полей на каждом графическом процессоре запускается основной цикл обработки лигандов. Общая схема вычислительного процесса с учётом использования нескольких графических процессоров представлена на рисунке 3.10.

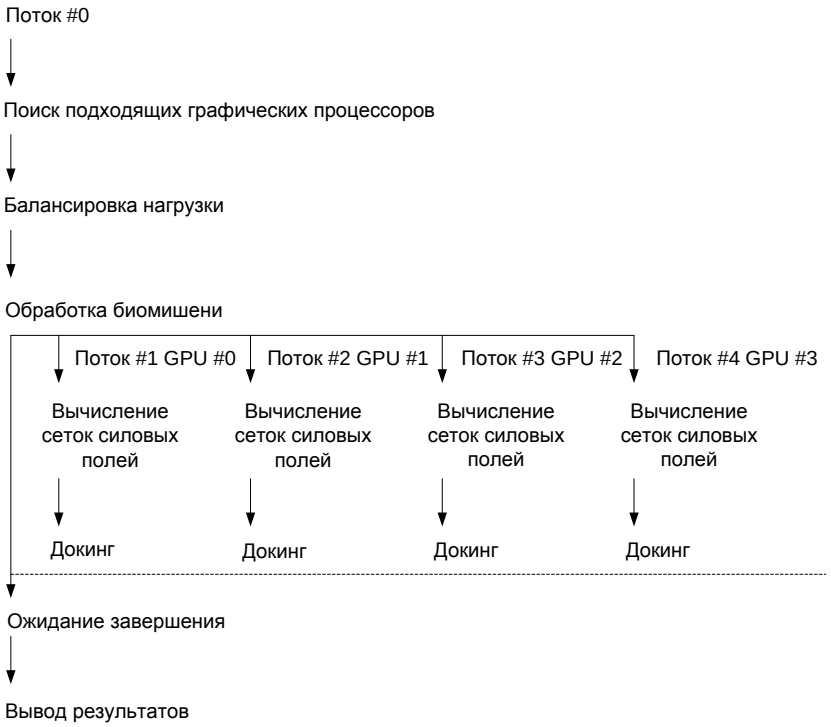


Рисунок 3.10 - Общая схема вычислительного процесса с учётом массива графических процессоров

3.5.4 Управление вычислениями на одном графическом процессоре

После получения списка лигандов для обработки и вычисления требуемых сеток силовых полей каждый поток, управляющий вычислениями на отдельном графическом процессоре, генерирует требуемое количество CUDA потоков (согласно количеству мультипроцессоров) и инициирует основной вычислительный цикл, в котором происходит выполнение докинга набора лигандов (псевдокод представлен на рисунке 3.11).

```
processed_ligands:=0;
portion_size:=PrepareNextPortion();
TranferDataToGpu();
while processed_ligands<total_ligands
        RunGpuCalculation();
        processed_ligands+=portion_size;
        portion_size:=PrepareNextPortion();
        Synchronize();
        TranferDataToGpu();
endwhile
```

Рисунок 3.11 - Псевдокод основного вычислительного цикла для одного графического процессора

Функция *PrepareNextPortion* подготавливает очередную группу лигандов: выполняет их чтение с диска, формирует вспомогательные структуры для представления лиганда на графическом процессоре, подготавливают кластеры векторов для метода дифференциальной эволюции. Кроме того, выделяется память под все необходимые структуры на графическом процессоре.

Функция*TranferDataToGpu* выполняет перенос информации об очередной порции лигандов и всех вспомогательных структур на графический процессор.

Функция *RunGpuCalculation* запускает вычисления на графическом процессоре. Вычислительное ядро DE асинхронно устанавливается в очередь каждого CUDA потока для каждого лиганда.

Функция *Synchronize* ожидает завершения вычислений во всех CUDA потоках, после чего выполняет перенос полученных результатов в память центрального процессора, отбор наилучших приближенных решений и освобождение памяти.

За счёт использования CUDA потоков и асинхронного запуска вычислительных ядер метода дифференциальной эволюции в процессе их вычисления подготавливается очередная группа лигандов, в результате к моменту завершения очередного этапа оптимизации новая итерация запускается незамедлительно.

Кластеры векторов метода дифференциальной эволюции инициализируются исходя из следующих соображений. Углы вращения всего лиганда вокруг координатных осей инициализируются при помощи генератора случайных числе в пределах 360° с шагом в 1°. Торсионные углы инициализируются при помощи генератора случайных чисел в пределах 10° от исходного значения с шагом в 1°. Для повышения точности моделирования, помимо исходных трёхмерных структур лигандов, на вход процедуры докинга также следует подавать конформеры, имеющие существенные отличия от исходных структур. По умолчанию смещения по координатным осямраспределяются равномерно, начиная от центра сетки, ограничивающей активный сайт связывания, что обусловлено изначальным предположением о

нахождении активного сайта биомишени, которое закладывается при позиционировании ограничивающей сетки.

3.5.5 Форматы файлов силового поля и обработка биомишени

Силовое поле описывается при помощи двух файлов параметров. Первый файл предназначен для описания аминокислотных остатков, входящих в белки, второй файл содержит в себе Ван-дер-Ваальсовы параметры для всех типов атомов, входящих в силовое поле.

Файл параметров представляет собой набор строк формата:

Файл = <Тип> {<Тип>}

Тип = <Имя_типа><Ван-дер-Ваальсов_радиус><Размерность_энергии><Гибридизация>

Гибридизация = 0 | 2 | 3.

Именем типа может быть любой набор цифр и букв латинского алфавита. Ван-дер-Ваальсов радиус и размерность энергии задаются вещественными числами и используются для вычисления энергии Ван-дер-Ваальсовых сил согласно (3.11).

Значение 2 для гибридизации означает sp2 гибридизацию, 3 означает sp3 гибридизацию, 0 означает, что значение гибридизации для атома неизвестно.

Файл с описанием параметров аминокислотных остатков в целом соответствует представлению силового поля AMBER за некоторыми изменениями. Файл содержит записи в следующем формате:

Список остатков = <Список_альтернатив><Остаток> {<Остаток>}

Список_альтернатив = **ALTE**<альтернатива> {<альтернатива>} **ENDALTE**

Альтернатива = <наименование_остатка>**=**<наименование_остатка>
{,<наименование_остатка>}

Остаток = **RESI**<наименование_остатка><атом> {<атом>} **ENDRESI**

Атом =
<порядковый_номер_в_остатке><PDB_обозначение_атома><FF_обозначение_атома><индекс_атома_связи><индекс_атома_валентного_угла><индекс_атома_торсионного_угла><длина_связи><валентный_угол><торсионный_угол><заряд><Тип_связи>

Тип_связи = 0 | 1 | 2.

Под наименованием остатка подразумевается его трёхбуквенное [135] обозначение (20 альфа-аминокислот имеют устоявшиеся обозначения, которые используются в рамках силового поля).

Часто при моделировании биомишени она может содержать в себе вариации стандартных аминокислотных остатков.Такие остатки должны быть заданы в силовом поле отдельно.При этом, если это необходимо, они могут быть указаны под обозначением стандартного остатка, но все его вариации должны быть записаны в список альтернатив, тогда они будут корректно распознаны в процессе разбора биомишени.

Под PDB обозначением атома подразумевается обозначение атома (обычно до трёх символов), принятое для него в PDB формате. Под FF обозначением атома подразумевается обозначение атома в рамках используемого силового поля.Обозначение должно присутствовать в файле параметровВан-дер-Ваальсовых сил. Под индексом атома связи подразумевается порядковый номер атома в остатке, с которым текущий атом ковалентно связан.При этом оптимальная длина связи задаётся параметром <длина_связи>. Индексом атома валентного угла задаётся порядковый номер атома, с которым текущий атом образует валентный угол через атом, заданный параметром <индекс_атома_связи>. Валентный угол задан параметром <валентный_угол>. Под индексом атома торсионного угла подразумевается порядковый номер атома, с которым текущий атом образует торсионный угол через атомы, заданные параметрами <индекс_атома_связи> и <индекс_атома_валентного_угла>. Образуемый торсионный угол задан параметром <торсионный_угол>. Параметр <тип_связи> задаёт тип ковалентной связи между текущий атомом и атомом, порядковый номер которого указан в <индекс_атома_связи>. Значение 1 соответствует одиночной связи, 2 – двойной, 0 – прочие ситуации. Пример описания для аминокислоты лизин представлен ниже (сама аминокислота изображена на рисунке 3.12).

```
RESI LYS
1  N    N     0  0  0    0.000    0.000    0.000   -.4937 1
2  H    H     1  0  0    1.010    0.000    0.000    .3018 1
3  CA   CT    1  0  0    1.449    0.000    0.000    .0343 1
4  HA   H1    3  1  0    1.090  109.500    0.000    .0464 1
5  CB   CT    3  1  0    1.525  111.100    0.000   -.0196 1
6  HB2  HC    5  3  1    1.090  109.500  300.000    .0143 1
7  HB3  HC    5  3  1    1.090  109.500   60.000    .0143 1
8  CG   CT    5  3  1    1.525  109.470  180.000   -.0233 1
9  HG2  HC    8  5  3    1.090  109.500  300.000    .0433 1
10 HG3  HC    8  5  3    1.090  109.500   60.000    .0433 1
11 CD   CT    8  5  3    1.525  109.470  180.000   -.0574 1
12 HD2  HC   11  8  5    1.090  109.500  300.000    .0602 1
13 HD3  HC   11  8  5    1.090  109.500   60.000    .0602 1
14 CE   CT   11  8  5    1.525  109.470  180.000    .1461 1
15 HE2  HP   14 11  8    1.090  109.500  300.000    .0470 1
16 HE3  HP   14 11  8    1.090  109.500   60.000    .0470 1
17 NZ   N3   14 11  8    1.470  109.470  180.000   -.2135 1
18 HZ1  H    17 14 11    1.010  109.470   60.000    .2872 1
19 HZ2  H    17 14 11    1.010  109.470  180.000    .2872 1
20 HZ3  H    17 14 11    1.010  109.470  300.000    .2872 1
21 CC     3  1  0    1.522  111.100    0.000    .6731 1
22 O    O    21  3  1    1.229  120.500    0.000   -.5854 2
ENDRESI
```

Из описания видно, что атом с обозначением CE (14) связан одиночной ковалентной связью с атомом CD (11), образует через него валентный угол с атомом CB(5) и торсионный угол (с осью, проходящей через атомы CB-CD) с атомом CA(3). Все связи являются одиночными. Длина ковалентной связи CE-CD равна 1,525 ангстрем. Валентный угол CE-CD-CB равен 109°47″. Торсионный угол CE-CD-CB-CA равен 180°. В файле силового поля указан заряд, умноженный на 18,22 для упрощения перевода в ккал/моль при выполнении вычислений.

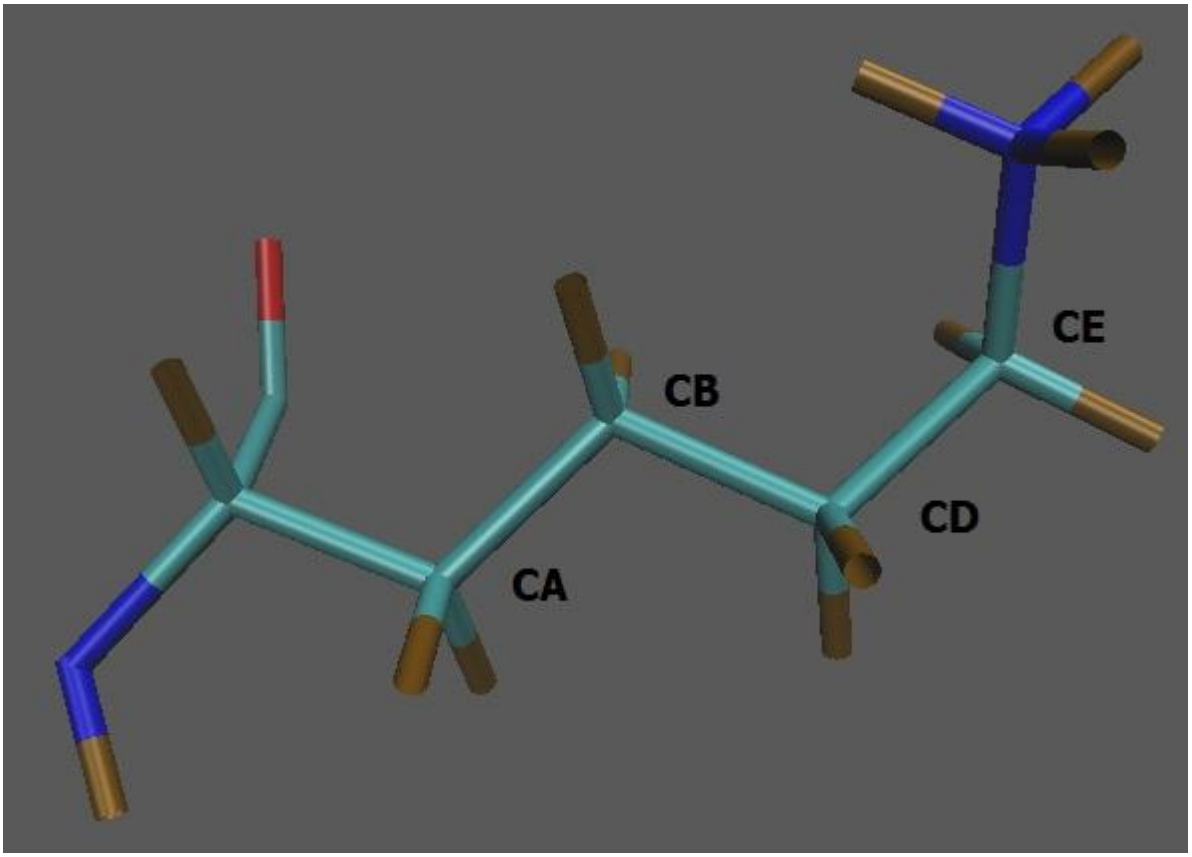


Рисунок 3.12 – Альфа-аминокислота Лизин

Каждая биомишень должна быть описана в рамках заданногосилового поля. Обозначения атомов в PDB файле должны соответствовать PDB обозначениям в силовом поле для установления соответствия. При разборе биомишени производится поиск и сравнение аминокислотных остатков, указанных в PDB файле и силовом поле. В результате чего каждому атому биомишени присваивается набор параметров: заряд, радиус Ван-дер-Ваальса, размерность энергии для потенциала Леннарда-Джонсона. Благодаря используемому параметру пользователи могут вносить любые изменения в силовое поле или полностью менять его согласно преследуемым целям.

3.5.6 Трёхмерная трансформация лиганда на графическом процессоре

На каждой итерации метода дифференциальной эволюции CUDA нити графического процессора выполняют трёхмерную трансформацию исходной

структуры лиганда согласно текущему приближенному решению (вектору из кластера). Для выполнения преобразований применяются матричные операции трёхмерной трансформации, максимальным образом скомпонованные для экономии вычислений. На первом этапе выполняется вращение по торсионным углам. В качестве оси вращения выступают два центральных атома торсионного угла. Ось вращения для каждого торсионного угла пересчитывается динамически согласно текущим координатам. После чего над каждым атомом, попавшим в список вращения по торсионному углу, выполняется преобразование, заданное матрицей вращения *RT* (3.12).

$\theta -$ угол поворота; $V(x, y, z) -$ ось вращения; $cs = \cos(\theta)$; $sn = \sin(\theta)$; $ic = 1 - cs$;

$$RT = \begin{bmatrix} x^2 \cdot ic + cs & x \cdot y \cdot ic + z \cdot sn & x \cdot z \cdot ic - y \cdot sn \\ x \cdot y \cdot ic - z \cdot sn & y^2 \cdot ic + cs & y \cdot z \cdot ic + x \cdot sn \\ x \cdot z \cdot ic + y \cdot sn & y \cdot z \cdot ic - x \cdot zn & z^2 \cdot ic + cs \end{bmatrix} \quad (3.12)$$

После выполнения вращений по торсионным углам выполняется поворот всего лиганта вокруг координатных осей. Матрица поворота всего лиганда формируется как комбинация матриц поворота вокруг каждой оси. Матрица поворота вокруг оси X представлена в (3.13).

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{bmatrix}, \quad (3.13)$$

где α – угол вращения вокруг оси X. Матрица поворота вокруг оси Y представлена в (3.14).

$$R_y = \begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) \\ 0 & 1 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) \end{bmatrix}, \quad (3.14)$$

где β – угол вращения вокруг оси Y. Матрица поворота вокруг оси Z представлена в (3.15).

$$R_z = \begin{bmatrix} \cos(\gamma) & -\sin(\gamma) & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad (3.15)$$

где γ – угол вращения вокруг оси Z. Итоговая матрица поворота будет выглядеть следующим образом:

$$R = \begin{bmatrix} \cos(\beta) \cdot \cos(\gamma) & \sin(\alpha) \cdot \sin(\beta) \cdot \cos(\gamma) - \cos(\alpha) \cdot \sin(\gamma) & \cos(\alpha) \cdot \sin(\beta) \cdot \cos(\gamma) + \sin(\alpha) \cdot \sin(\gamma) \\ \cos(\beta) \cdot \sin(\gamma) & \sin(\alpha) \cdot \sin(\beta) \cdot \sin(\gamma) + \cos(\alpha) \cdot \cos(\gamma) & \cos(\alpha) \cdot \sin(\beta) \cdot \sin(\gamma) - \sin(\alpha) \cdot \cos(\gamma) \\ -\sin(\beta) & \sin(\alpha) \cdot \cos(\beta) & \cos(\alpha) \cdot \cos(\beta) \end{bmatrix} \quad (3.16)$$

После выполнения всех этапов трансформации лиганда выполняется проверка на выход его атомов за границы сетки, а также проверка на поворот «в себя». В случае если в результате трансформации образуется нефизическая структура лиганда, вычисление энергии не производится, и на очередном этапе в качестве значения энергии данному приближенному решению присваивается пенальти — некоторое большое положительное число, гарантирующее, что вектор не будет закреплён и не перейдёт на следующую итерацию метода.

В случае если структура признана возможной, производится вычисление её энергии. Энергия вычисляется как линейная комбинация энергий всех атомов лиганда. Компоненты энергии атома вычисляются исходя из сеток силовых полей на основе его заряда и типа. Чтобы на основе сеток силового поля получить значение энергии для произвольного расположения атома, выполняется трилинейная интерполяция, которая в общем виде задаётся как:

$$
\begin{aligned}
f(x,y,z) = {} & \frac{f(x_1,y_1,z_1)}{(x_2-x_1)(y_2-y_1)(z_2-z_1)}(x_2-x)(y_2-y)(z_2-z)+ \\
& \frac{f(x_1,y_1,z_2)}{(x_2-x_1)(y_2-y_1)(z_2-z_1)}(x_2-x)(y_2-y)(z-z_1)+ \\
& \frac{f(x_1,y_2,z_1)}{(x_2-x_1)(y_2-y_1)(z_2-z_1)}(x_2-x)(y-y_1)(z_2-z)+ \\
& \frac{f(x_1,y_2,z_2)}{(x_2-x_1)(y_2-y_1)(z_2-z_1)}(x_2-x)(y-y_1)(z-z_1)+ \\
& \frac{f(x_2,y_1,z_1)}{(x_2-x_1)(y_2-y_1)(z_2-z_1)}(x-x_1)(y_2-y)(z_2-z)+ \\
& \frac{f(x_2,y_1,z_2)}{(x_2-x_1)(y_2-y_1)(z_2-z_1)}(x-x_1)(y_2-y)(z-z_1)+ \\
& \frac{f(x_2,y_2,z_1)}{(x_2-x_1)(y_2-y_1)(z_2-z_1)}(x-x_1)(y-y_1)(z_2-z)+ \\
& \frac{f(x_2,y_2,z_2)}{(x_2-x_1)(y_2-y_1)(z_2-z_1)}(x-x_1)(y-y_1)(z-z_1),
\end{aligned}
\tag{3.17}
$$

где x,y,z – координаты атома, для которого необходимо вычислить энергию; $f(x_i,y_i,z_i)$ – значения компонент энергии для ячеек сетки; x_i,y_i,z_i– координаты ячеек сетки. Иллюстрация интерполяции представлена на рисунке 3.13.

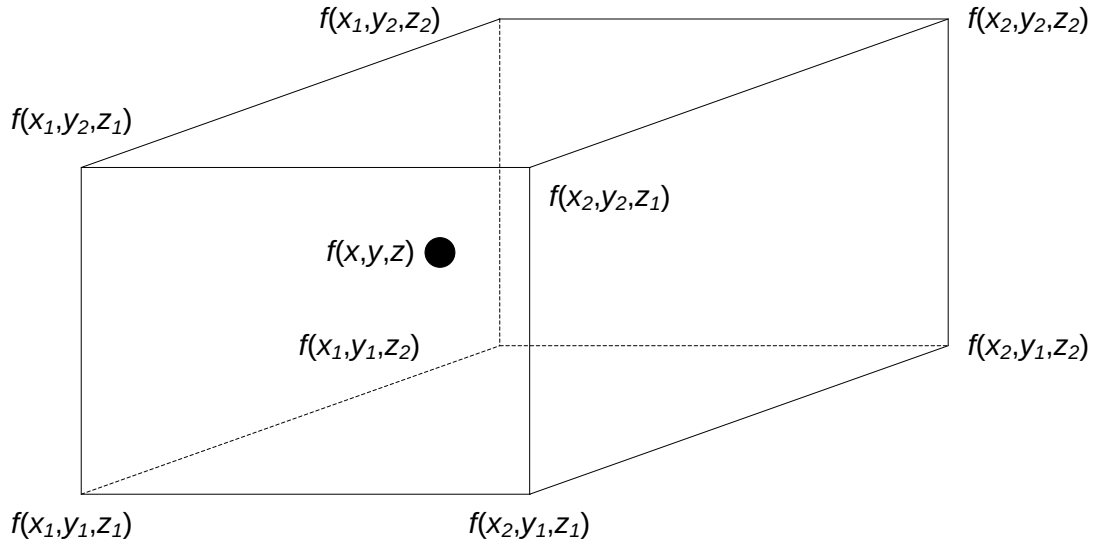


Рисунок 3.13 - Иллюстрация трилинейной интерполяции для сетки силового поля

Так как шаг сетки фиксирован, из формулы (3.17) во всех частичных суммах может быть вынесен знаменатель дроби. В результате итоговая формула для выполнения интерполяции будет выглядеть следующим образом:

$$
\begin{aligned}
f(x,y,z) = (&f(x_1,y_1,z_1)(x_2-x)(y_2-y)(z_2-z)\\
&+ f(x_1,y_1,z_2)(x_2-x)(y_2-y)(z-z_1)\\
&+ f(x_1,y_2,z_1)(x_2-x)(y-y_1)(z_2-z)\\
&+ f(x_1,y_2,z_2)(x_2-x)(y-y_1)(z-z_1)\\
&+ f(x_2,y_1,z_1)(x-x_1)(y_2-y)(z_2-z)\\
&+ f(x_2,y_1,z_2)(x-x_1)(y_2-y)(z-z_1)\\
&+ f(x_2,y_2,z_1)(x-x_1)(y-y_1)(z_2-z)\\
&+ f(x_2,y_2,z_2)(x-x_1)(y-y_1)(z-z_1))/grid\_step^3
\end{aligned}
\tag{3.18}
$$

3.6 Выводы по третьей главе

1. Предложена программная архитектура для вычисления сеток силовых полей с использованием графических процессоров, позволяющая эффективно автоматически распределять вычисления для большого набора биомишений между всеми доступными ресурсами как одного, так и нескольких графических процессоров. Кроме того, предложенная реализация позволяет автоматически комбинировать вычисления сеток силовых полей для схожих Ван-дер-Ваальсовых параметров атомов проб, независимо от используемого силового поля, что позволяет уменьшить количество необходимых вычислений.

2. Предложеналгоритм менеджера времени выполнения, который позволяет дополнительно уменьшить задержки работы с медленной дисковой памятью, за счёт многопоточного управления очередями задач вычисления различных сеток силовых полей. Менеджер времени выполнения может работать в вычислительных системах, укомплектованных несколькими графическими процессорами.

3. Предложенные алгоритмы реализация метода дифференциальной эволюции и вычисления сеток силовых полей интегрированы в алгоритм выполнения молекулярного лиганд-белкового докингас использованием графических процессоров. Предложенная реализация позволяет ускорить выполнение молекулярного лиганд-белкового докинга приобработке больших баз

лигандов. Программное обеспечение автоматически распределяет вычислительную нагрузку по всем доступным графическим процессорам системы, использует точное силовое поле и учитывает основные компоненты межмолекулярного взаимодействия.

Результаты, представленные в главе, опубликованы в работах [73-76,85].

4 Анализ эффективности разработанных алгоритмов и программ

В главе представлены детали и результаты процедур тестирования предлагаемых решений. Проведены сравнения скорости выполнения предлагаемых алгоритма реализации метода дифференциальной эволюции, алгоритмавычисления сеток силовых полей, алгоритма решения задачи лиганд-белкового молекулярного докинга по сравнению с существующими подходами.

4.1 Тестирование алгоритма реализации метода дифференциальной эволюции

Предложенная реализация на графических процессорах метода дифференциальной эволюции была протестирована на предмет работоспособности и скорости сходимости на наборе тестовых задач оптимизации [77, 114, 115]. В качестве эталонных значений по скорости сходимости были взяты данные из оригинальной работы по методу дифференциальной эволюции [77]. Ряд многомерных функции был протестирован при различных размерах векторов оптимизируемых параметров. Результаты тестов для всех функций представлены в таблице 4.1. Размер популяции (NP) для метода дифференциальной эволюции выбирался как максимальная граница рекомендованного интервала для текущего количества оптимизируемых параметров (D). Параметры метода дифференциальной эволюции (F и CR), а также контрольное значение (VTR) и область инициации параметров функции (IPR) были взяты из [77] без изменения. Для каждой функции была выполнена процедура оптимизации и получено количество вычислений функции (NEF). По результатам тестов следует отметить, что для всех функций предлагаемая реализация метода, использующая графические процессоры, сошлась к глобальному экстремуму и почти для всех функций показала требуемую скорость сходимости. Исключение составила лишь функция Экли для 100

оптимизируемыхпеременных, где скорость сходимости оказалась ниже (81154 NEF против заявленных 36801[77]).

Таблица 4.1 – Результаты тестирования сходимости алгоритма реализации метода дифференциальной эволюции на графических процессорах

Функция	D	NP	F	CR	VTR	IPR	NEF
Goldstein's function	1	10	0,5	0,1	7,1	[-10;10]	18
De Jong function 1	3	30	0,9	0,1	e-6	[-5,12;5,12]	67
Rosenbrock's saddle	2	20	0,9	0,9	e-6	[-2,048;2,048]	60
Penalized Shubert Function	1	10	0,5	0	-12,8	[-10;10]	26
2d Penalized Shubert Function	2	20	0,5	0	-186,73	[-10;10]	148
Hyper-Ellipsoid	30	30	0,5	0,1	e-10	[-1;1]	481
Hyper-Ellipsoid	100	1000	0,5	0,1	e-10	[-1;1]	3204
Griewangk's function	20	200	0,5	0,2	e-3	[-600;600]	705
Griewangk's function	100	1000	0,5	0,2	e-3	[-600;600]	2216
Rastrigin's Function	20	200	0,5	0	0,9	[-600;600]	329
Rastrigin's Function	100	1000	0,5	0	0,9	[-600;600]	3849
Ackley's function	30	300	0,5	0,1	e-3	[-30;30]	1580
Ackley's function	100	1000	0,5	0,1	e-3	[-30;30]	81154

Для тестирования производительности предложенного решения была выбрана функция Экли как наиболее вычислительно затратная процедура в приведённом тестовом наборе. В качестве параметров для тестов были установлены: размер вектора параметров 100, размер кластера векторов 100. Параметры для метода дифференциальной эволюции были оставлены без изменения. При тестировании была исключена остановка процедуры оптимизации при достижении экстремума с целью исключить искажения в оценке производительности. Все тесты выполнялись для 10000 NEF. В качестве тестовой платформы использовался сервер, укомплектованный двумя процессорами IntelXeonE5-2640 2,5ГГц (12 вычислительных ядер), 256 ГБ оперативной памяти и графическим процессором NVIDIATeslaK20m, включающим в себя 13 мультипроцессоров (MX) по 192 CUDA ядер каждый. Тестирование на центральном процессоре производилось в двух вариантах: для одного потока и для 12 потоков. Тестирование на графическом процессоре производилось для одного мультипроцессора. Результаты тестов представлены в таблице 4.2. Тестирование продемонстрировало, что на тестовой задаче предлагаемое решение демонстрирует ускорение в 2 раза по сравнению с 12ти потоковой реализацией,

таким образом при полной загрузке тестового графического процессора обеспечивая ускорение до 26 раз.

Таблица 4.2 – Результаты тестирования производительности алгоритма реализации метода дифференциальной эволюции на графических процессорах

Конфигурация	Время вычислений, сек	Ускорение	Ускорение в расчёте на общее количество MX
CPU 1 thread	951	6,75	86,46
CPU 12 threads	287	2,04	26,09
GPU 1 MX*	141	-	-

* предлагаемое решение

Вместе с тем отдельно следует отметить высокую универсальность метода дифференциальной эволюции. С момента появления метод успешно применяется в различных прикладных областях: рентгенография и кристаллография [128, 129], телекоммуникации [130], анализ изображений [131], электромагнетизм [132], обработка сигналов [133], проектирование солнечных батарей [134].

4.2 Тестирование алгоритма вычисления сеток силовых полей

Для выполнения тестов было отобрано несколько графических процессоров, имеющих различный уровень вычислительных возможностей. Краткая информация о графических процессорах для тестов представлена в таблице 4.3.

Таблица 4.3 – Набор тестовых графических процессоров

Графический процессор	Уровень вычислительных возможностей	Кол-во мультипроцессоров	Кол-во CUDA ядер
TeslaM2090	2.0	16	512
Tesla K20m	3.0	13	2496
GeForce GT 440	2.1	2	96

Предложенная реализация сравнивалась с возможностями программы AutoGrid 4. Остальные решения, рассмотренные выше, не предоставляют исходных код или скомпилированное программное обеспечение. В качестве теста использовалось вычисление 9 сеток силовых полей для одной биомишени. Это типовое значение для программы AutoGrid 4. Размер сетки был выбран равным 64, шаг – 0,3 ангстрема по всем измерениям. AutoGrid 4 исполнялся на

компьютере,укомплектованном центральным процессором IntelCorei7 920 с тактовой частотой 2,3 ГГц и 3ГБ оперативной памяти.

Результаты тестов представлены в таблице 4.4, из которой видно, что даже на графическом процессоре GeForceGT440, имеющем сравнимо малое количество CUDA ядер, удалось достигнуть значительного ускорения.

Таблица 4.4 – Среднее время вычисления 9 сеток силовых полей

Конфигурация	Затраченное время, мс	Ускорение
AutoGrid 4	74010	-
Tesla M2090*	1824,04	40,58
Tesla K20m*	1523,21	48,59
GeForce GT 440*	11370,01	6,51

* предлагаемое решение

Ускорение за счёт использования предложенной модели менеджера времени выполнения тестировалось с применением графического процессора TeslaM2090. Было выполнено три теста, в каждом из которых производилось вычисление 60 сеток. В первом случае выполнялось вычисление сеток размерностью 64 по всем измерениям, во втором – 128, в третьем размер сеток менялся от 32 до 128. Результаты тестов представлены в таблице 4.5, из которой видно, что удалось достигнуть дополнительного ускорения за счёт предложенного менеджера времени выполнения.

Таблица 4.5 – Результаты тестов однопотокового управления вычислениями и управления вычислениями с использованием предложенного менеджера времени выполнения (столбец i_o_d)

Тест	Один поток,с.	i_o_d,с.	Ускорение
64*64*64	221	145	1,52
128*128*128	1296	877	1,48
Различный размер (32-128)	702	334	2,11

Кроме того, было произведено тестирование равномерности балансировки нагрузки между несколькими графическими процессорами. Было выбрано два массива графических процессоров: два процессора TeslaM2090 и четыре процессора TeslaK20m. Следует отметить, что это наиболее распространённые

конфигурации многопроцессорных массивов. Результаты тестов для массива из двух графических процессоров TeslaM2090 представлены в таблице 4.6.

Таблица 4.6 – Время работы потока d_thread и потока o_thread для двух процессоров TeslaM2090

Тест	d_thread, c.		o_thread, c.	
	#0	#1	#0	#1
64*64*64	124	128	114	117
128*128*128	761	754	620	585
Различный размер (32-128)	235	237	290	291

Результаты тестов для массива из четырёх графических процессоров TeslaK20m представлены в таблице 4.7.

Таблица 4.7 – Время работы потока d_thread и потока o_thread для четырёх процессоров TeslaK20

Тест	d_thread,c.				o_thread, c.			
	#0	#1	#2	#3	#0	#1	#2	#3
64*64*64	41	43	45	46	69	73	74	76
128*128*128	264	256	267	255	459	421	470	442
Различный размер (32-128)	96	86	105	95	165	136	163	159

Из результатов тестов на балансировку видно, что нагрузка распределяется равномерно даже для сеток, имеющих различный размер.

4.3 Тестирование алгоритма решения задачи молекулярного лиганд-белкового докинга

Для тестирования предложенной реализации использовались графические процессоры NVIDIATeslaK20m и TeslaK40. Тесты проводились как для одиночных графических процессоров, так и для массива графических процессоров. Краткая информация о тестовом наборе представлена в таблице 4.8.

Таблица 4.8 – Набор графических процессоров для выполнения молекулярного докинга

Графический процессор	Уровень вычислительных возможностей	Кол-во мультипроцессоров	Кол-во CUDA ядер
Tesla K40	3.5	16	3072
Tesla K20m	3.0	13	2496

В качестве альтернативных решений, использующих возможности только центрального процессора, были выбраны наиболее популярные пакеты для выполнения молекулярного докинга: AutoDock4 (как наиболее точное существующее решение) и AutoDockVina (как наиболее быстрое существующее решение). Тестирование возможностей отобранных пакетов производилось на сервере, укомплектованном двумя процессорами IntelXeonE5-2640 2,5ГГц, содержащими 12 ядер каждый, что обеспечивает до 24 вычислительных потоков в режиме HyperThreading. Сервер содержит 256 ГБ оперативной памяти.

Прежде всего были оценены временные затраты работы программ в пересчёте на один лиганд, что необходимо для последующих тестов скрининга большой базы данных. В качестве биомишени использовалась молекула с PDB идентификатором 1ULB, в качестве лиганда использовалась молекула с ZINC идентификатором 895129. Программа AutoDockVinaтестировалась в трёх конфигурациях, так как использует возможности многоядерных систем и позволяет управлять количеством используемых потоков. В первом случае, количество используемых потоков было установлено в 24 для полной загрузки всех доступных ресурсов.Во втором случае, количество потоков было установлено в 12, что эквивалентно количеству ядер на тестовом компьютере.В третьем случае, количество потоков было установлено в 6 (что эквивалентно количеству ядер одного процессора сервера) для исключения влияния межпроцессорного взаимодействия. Для программы AutoDock4 время, затрачиваемое на подготовкуи предварительное вычисление сеток силовых полей, в результаты тестов не включалось. Для предлагаемой реализации производился запуск с копиями тестового лиганда в количестве, эквивалентном количеству мультипроцессоров графического процессора, после чего время делилось на это количество. Результаты тестов представлены в таблице 4.9.

Таблица 4.9 – Время выполнения молекулярного докинга в расчете на один лиганд

Программа	Время выполнения, с
AutoDock Vina (cpu=24)	16
AutoDock Vina (cpu=12)	15,9
AutoDock Vina (cpu=6)	19,5
AutoDock4	153,2
GDock NVIDIA Tesla K20m*	3,12
GDock NVIDIA Tesla K40m*	1,27

* предлагаемое решение

По результатам проведённого теста видно, что предлагаемое решение работает быстрее в пересчёте на один лиганд. Кроме того, несмотря на высокую точность работы, программа AutoDock4 была исключена из теста скрининга базы данных лигандов из-за чрезвычайно низкой скорости работы.

Для выполнения скрининга была выбрана биомишень с PDB идентификатором 3PTB и лиганд с ZINC идентификатором 36634. К тестовому набору случайным образом было добавлено 99 лигандов, содержащихся в открытой базе данных химических соединений (лекарств, разрешённых к использованию). Программа AutoDockVina тестировалась в ранее описанных трёх конфигурациях. Предлагаемое решение дополнительно тестировалось на массиве из трёх графических процессоров, для каждого типа отдельно. Результаты тестов представлены в таблице 4.10.

Таблица 4.10 – Время выполнения молекулярного докинга для набора из 100 лигандов

Программа	Время выполнения, с
AutoDock Vina (cpu=24)	2253
AutoDock Vina (cpu=12)	2239
AutoDock Vina (cpu=6)	2302
GDock NVIDIA Tesla K20m*	322
GDock NVIDIA Tesla K20m X3*	85
GDock NVIDIA Tesla K40*	131

* предлагаемое решение

Таким образом, по результатам тестов видно, что предлагаемое решение демонстрирует существенный прирост производительности до 17 раз при использовании одного графического процессора и до 27 раз при использовании

массива графических процессоров даже по сравнению с многопоточными решениями.

Для тестирования точности вычисления был выполнен молекулярный докинг для ряда биомишений с их лигандами, позиционирование которых в активном сайте было определено при помощи кристаллографии. По результатам проведения процедуры, для лучшей структуры, отобранной предлагаемым решением, было посчитано среднеквадратичное отклонение (RMSD) предсказанного положения от истинного. Результаты тестов представлены в таблице 4.11.

Таблица 4.11 – Среднеквадратичное отклонение предсказанных конформаций молекулы биомишени и лиганда от истинного значения

Комплекс	RMSD,ангстрем
PDB:1ULB	2,15
PDB:2MCP	1,79
PDB:3PTB	0,44

Трёхмерные структуры и результаты их позиционирования представлены на рисунке 4.1.

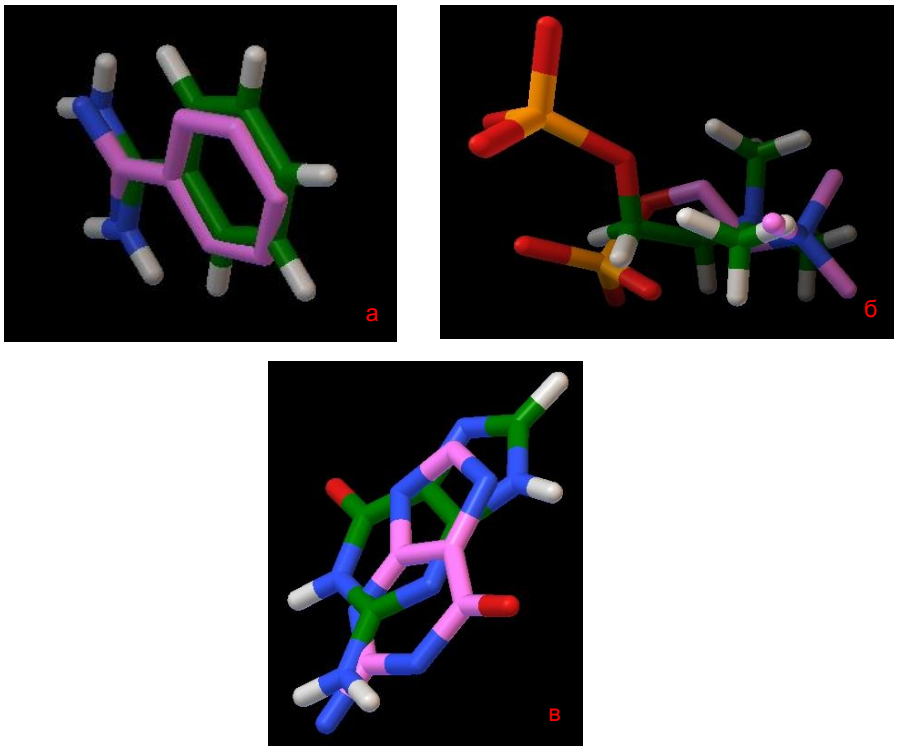


Рисунок 4.1 - Результаты предсказания трёхмерных структур. а) Структура 3PTB; б) Структура 2MCP; в) Структура 1ULB.

Таким образом, по результатам серии тестов можно сделать выводы, что предлагаемые решения значительно быстрее существующих аналогов, использующих только возможности центрального процессора. Предложенный алгоритм выполнения молекулярного лиганд-белковго докинга также демонстрируют приемлемую точность, несмотря на то, что в процессе выполнения используется только глобальный поиск.

4.4 Выводы по четвёртой главе

1. Разработанный алгоритм вычисления сеток силовых полей демонстрирует ускорение до 49 раз по сравнению с существующими решениями.

2. Предложенный менеджер времени выполнения для вычисления сеток силовых полей позволяет получить дополнительное ускорение до 2 раз.

3. Предложенный алгоритм выполнения молекулярного лиганд-белкового докинга обеспечивает ускорение до 12 раз в расчёте на один лиганд по сравнению с аналогичными решениями, использующими до 24 ядер центрального процессора.

4. Предложенный алгоритм выполнения молекулярного лиганд-белкового докинга обеспечивает ускорение до17 раз при скрининге баз данных лигандов по сравнению с аналогичными решениями, использующими до 24 ядер центрального процессора, и до 27 раз при использовании массива графических процессоров.

5. Предложенный алгоритм на реальных химических соединениях демонстрирует точность выполнения докинга в пределах 2,5 ангстрема.

ЗАКЛЮЧЕНИЕ

В рамках диссертации изучались методы и алгоритмы применения численных методов оптимизации на графических процессорах для решения прикладных задач и их применение для эффективного решения задачи молекулярного лиганд-белкового докинга.

К основным результатам диссертационного исследования следует отнести:

1. Предложен алгоритм реализации метода дифференциальной эволюции с использованием графических процессоров. Предлагаемая композиция этапов дифференциальной эволюции обеспечивает распределение отдельных процедур оптимизации по мультипроцессорам графического процесса, удовлетворяя требованиям метода дифференциальной эволюции к размеру популяции, позволяя тем самым увеличить нагрузку на графический процессор и использовать встроенные средства синхронизации. Разработанный подход хорошо подходит для решения задач, связанных с выполнением большого количества процедур оптимизации.

2. Разработан алгоритм расчёта силовых полей межмолекулярного взаимодействия на основе вычисления сеток с использованием графических процессоров. Предлагаемый метод выполнения позволяет эффективно автоматически распределять вычисления различных типов сеток силовых полей для больших баз химический соединений между доступными ресурсами графических процессоров. Управление вычислениями организовано таким образом, чтобы обеспечить максимальную загрузку графических процессоров и скрыть задержки работы с дисковой памятью. Алгоритм демонстрирует значительное ускорение вычислений по сравнению с аналогами.

3. Предложен алгоритмрешения задачи молекулярного докинга с использованием графических процессоров. Разработанное решение ориентировано на автоматическую обработку больших баз лигандов и позволяет прозрачно для пользователя распределять нагрузку по всем доступным графическим процессорам. Предлагаемый подход демонстрирует существенное

ускорение моделирования и требуемую точность вычислений при обработке реальных химических соединений.

4. Создан комплекс программного обеспечения, реализующий предложенные алгоритмы.Получены свидетельства о регистрации программ для ЭВМ в Федеральном институте промышленной собственности №2013660687, №2014619771.

Предложенные алгоритмы и программные решения создают научную основу решения важных прикладных задач фармакологии и биоинформатики на базе гетерогенных вычислительных систем, использующих графические процессоры. Данные средства могут использоваться как самостоятельно, так и применяться для реализации новых подходов к решению этих задач.

ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ

CUDA (*ComputeUnifiedDeviceArchitecture*)**:** Программно-аппаратная массивно-параллельная архитектура, используемая в графических процессорах компании NVIDIA.

CUDA ядро: Арифметико-логическое устройство, входящее в состав мультипроцессора графического процессора в большом количестве.

invitro: Эксперименты «в стекле», т.е. в пробирке.

invivo: Эксперименты в живом организме.

insilico: Компьютерное моделирование.

MPP (*MassiveParallelProcessing*): Архитектура параллельных вычислительных систем, суперкомпьютеров Как правило, отличительной особенностью от кластерных вычислительных систем является применение специфических технологических решений при построении компонент системы.

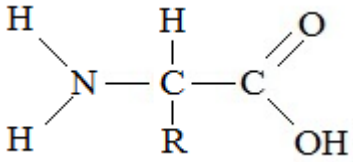
sp2 гибридизация: Гибридизация одной s орбитали и двух p орбиталей. В результате образуется три гибридных орбитали, расположенные в одной плоскости с углом в 120°.

sp3 гибридизация: Гибридизация одной s орбитали и трёх p орбиталей. В результате образуется четыре гибридных орбитали, оси которых направлены к вершинам правильного тетраэдра.

варп (warp): Группа нитейграфического процессора, которая выполняется одновременно на одном мультипроцессоре.

аминогруппа: Функциональная группа атомов –NH2.

аминокислоты: Органические соединения, молекула которых состоит из аминогруппы, карбоксильной группы и углеводородного радикала. Подразделяются на: альфа-, бета-, гамма- и т.д. аминокислоты в зависимости от места присоединения аминогруппы. В природе встречаются только альфа-аминокислоты.



ангстрем: Единица измерения расстояния. 1 ангстрем равен 0,1 нм.

биомишень: Большое молекулярное соединение (как правило, белок), насчитывающее несколько сотен атомов и являющееся целью получение биологического отклика.

валентность: Свойство атомов химических элементов образовывать химические связи.

вычислительное ядро: Программная единица, функция, исполняющаяся на графическом процессоре.

геометрические изомеры: Изомеры, имеющие различное пространственное расположение атомов в молекуле и не относящиеся к оптическим изомерам. Подразделяются на цисизомеры и трансизомеры.

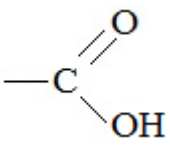
гетерогенная вычислительная система: Вычислительная система, в состав которой входят вычислительные устройства различных архитектур.

гибридизация: Это взаимодействие различных по типу, но близких по энергии атомных орбиталей с образованием гибридых орбиталей одинаковой формы и энергии.

гидроксильная группа: Функциональная группа атомов –OH.

ингибитор: Вещество, замедляющее течение какой-либо химической реакции.

изомеры: Химические соединения, имеющие одинаковый качественный и количественный состав, но разное строение и, как следствие, разные свойства.

карбоксильная группа: Функциональная группа атомов –COOH. 

ковалентная связь: Химическая связь, образованная перекрытием пары валентных электронных облаков.

конформация молекулы: Пространственное расположение атомов в молекуле при вращении атомов или групп атомов вокруг простых связей при сохранении неизменным порядка химических связей атомов.

конформер: Стереоизомер в ряде конформаций, соответствующий минимуму потенциальной энергии.

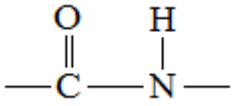
лиганд: Небольшое молекулярное соединение (как правило, кандидат в лекарственный препарат), насчитывающее несколько десятков атомов.

межмолекулярное взаимодействие: Взаимодействие молекул между собой, которое не приводит к образованию новых или разрыву старых химических связей.

молекулярный докинг: Это метод компьютерного моделирования межмолекулярных взаимодействий.

молекулярная формула: Формула, отражающая качественный состав соединения.

орбиталь: Пространство вокруг ядра атома, в котором заключено 90% электронного облака.

пептидная связь: Это связь, образующаяся между остатком –NH– аминогруппы одной молекулы аминокислоты и остатком –CO– карбоксильной группы другой молекулы аминокислоты.  .

программируемая логическая интегральная схема (ПЛИС): Интегральная схема, логика работы не задана жёстко в процессе производства и может изменять программным образом.

ротамер: Частный случай конформера, являющийся результатом вращения группы атомов вокруг одной химической связи.

сайт связывания (*bindingsite*): Область поверхности молекулы биомишени, к которой происходит присоединения молекулы лиганда.

энергия связывания, свободная энергия связывания: Это величина, показывающая изменение энергии в ходе химической реакции.

спин: Собственный магнитный момент электрона.

структурная формула: Формула, отражающая порядок соединения атомов в молекуле.

ферменты, энзимы: Органические катализаторы химических реакций в живых организмах.

функциональная группа: Группа атомов, которая определяет наиболее характерные свойства вещества и его принадлежность к определённому классу соединений (например, гидроксильная группа –OH определяет принадлежность к алканолам).

шейдер: Программа для управления вычислениями на определённой стадии обработки графической информации. Различают: вершинные, пиксельные и геометрические шейдеры.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Kuntz, I. Structure-based strategies for drug design and discovery / I. Kuntz // Science.– 1992.–№ 257.–С. 1078.

2 Sukhwani, B. Accelerating molecular docking and binding site mapping using FPGAs and GPUs. Ph.D. thesis. / B. Sukhwani.–BostonUniversity. CollegeofEngineering.– 2011.– 286 с.

3 Фарков, М.А. Выполнениемолекулярногодокингасиспользованиемграфическихускорителей / М.А. Фарков // СборниктрудовXIIМеждународной молодёжной научной конференции «Интеллект и наука труды».– Красноярск.– 2012.– С. 181.

4 Pechan, I. Implementing a Global Optimization Algorithm Related To Bioinformatics with a High-Performance FPGA / I. Pechan // In Proceedings of the Sixteenth PhD Mini-Symposium, Budapest University of Technology and Economics.– 2009.–С. 60.

5 Sukhwani, B. GPU Acceleration of a Production Molecular Docking Code / B. Sukhwani, M.C. Herbordt // International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS). GPGPU.– 2009.–С. 19.

6 Ritchie, D.W. Ultra-fast FFT protein docking on graphics processors / D.W. Ritchie, V. Venkatraman // Bioinformatics.– 2010.–Т. 26, № 19.–С. 2398.

7 Pechan, I. Hardware Accelerated Molecular Docking: A Survey [Электронныйресурс] / I. Pechan, B.Fehér // Bioinformatics.– 2012.– Режим доступа:http://www.intechopen.com/books/bioinformatics/hardware-accelerated-molecular-docking-a-survey.

8 Korb, O. Accelerating molecular docking calculations using graphics processing units / O. Korb, T. Stützle, T.E. Exner // Journal of chemical information and modeling. American Chemical Society.– 2011.–Т. 51, № 4.–С. 865.

9 Simonsen, M. GPU-accelerated high-accuracy molecular docking using guided differential evolution / M. Simonsen [идр.] // Proceeding GECCO ’11.– 2011.– С. 1803.

10 Kannan, S. Porting Autodock to CUDA / S. Kannan, R. Ganji// Proceeding WCCI 2010 IEEE World Congress on Computational Intelligence.–2010.–С. 1.

11 RCSBProteinDataBank [Электронныйресурс].– RCSBProteinDataBank.– Режимдоступа: http://www.pdb.org/pdb/home/home.do.

12 ProteinDataBankJapan [Электронныйресурс].– ProteinDataBankJapan.– Режимдоступа: http://www.pdbj.org.

13 ProteinDataBankEurope [Электронныйресурс].– ProteinDataBankEurope.– Режимдоступа: http://www.ebi.ac.uk/pdbe.

14 PDBbind-CNDatabase[Электронныйресурс].– PDBbind-CNDatabase.– Режимдоступа: http://www.pdbbind-cn.org.

15 Wang, R. The PDBbind Database: Methodologies and updates"/ R. Wang [идр.] // Journal of Medicinal Chemistry.– 2005.–Т. 48, №12.–С. 4111.

16 Wang, R.The PDBbind Database: Collection of Binding Affinities for Protein-Ligand Complexes with Known Three-Dimensional Structures / R. Wang [идр.] // Journal of Medicinal Chemistry.– 2004.–Т. 47, № 12.–С. 2977.

17 Protein-Protein Docking Benchmark [Электронныйресурс].–Protein-Protein Docking Benchmark.– Режимдоступа: http://zlab.umassmed.edu/benchmark.

18 Chen, R. A Protein-Protein Docking Benchmark / R. Chen [идр.] //Proteins.–2003.– Т. 53, № 1.– С. 88.

19 ZINCDatabase[Электронныйресурс].– ZINCDatabase.– Режимдоступа: http://zinc.docking.org.

20 Зефиров, Н.С. Современныепринципыконструирования лекарств / Н.С. Зефиров // Вестник Российской Академии Наук.– 2004.– Т. 74, № 5.–С. 418.

21 Koshland, D. Jr. The joys and vicissitudes of protein science / D. Jr. Koshland // Protein Science.– 1993.– № 2.–С. 1364.

22 Koshland, D. Jr. Application of a theory of enzyme specificity to protein synthesis / D. Jr. Koshland // Proceedings of the National Academy of Sciences.– 1958.– № 44.–С. 98.

23 Kozakov, D. PIPER: An FFT-Based Protein Docking Program with Pairwise Potentials / D. Kozakov [идр.] //PROTEINS: Structure, Function, and Bioinformatics.– 2006.– № 65.–C. 392.

24 Rong, C. Docking Unbound Proteins Using Shape Complementarity, Desolvation, and Electrostatics / C. Rong, W. Zhiping //PROTEINS: Structure, Function, and Genetics.– 2002.– № 47.– C.281.

25 Kuntz, I. A geometric approach to macromolecule-ligand interactions / I. Kuntz [идр.] // Journal of Molecular Biology.– 1982.– №.161.–C. 269.

26 Trott,O. Software News and Update AutoDock Vina: Improving the Speed and Accuracy of Docking with a New Scoring Function, Efficient Optimization, and Multithreading / O. Trott, A. Olson //Journal Computational Chemistry.– 2010.– № 31.– C.455.

27 Huey, R. A Semiempirical Free Energy Force Field with Charge-Based Desolvation / R. Huey [идр.] //Journal Computational Chemistry.– 2007.– № 28.– C. 1145.

28 Todd, J. DOCK 4.0: Search strategies for automated molecular docking of flexible molecule databases / J. Todd [идр.] // Journal of Computer-Aided Molecular Design.– 2001.– № 15.– C. 411.

29 Liu, M. MCDOCK: A Monte Carlo simulation approach to the molecular docking problem / M. Liu, S. Wang // Journal of Computer-Aided Molecular Design.– 1999.– № 13.– C. 435.

30 Morris, G.M. AutoDock4 and AutoDockTools4: Automated docking with selective receptor flexibility / G.M. Morris [идр.] //Journal Computational Chemistry.– 2009.– № 30.– C. 2785.

31 Jain, A.N. Surflex: Fully Automatic Flexible Molecular Docking Using a Molecular Similarity-Based Search Engine / A.N. Jain // Journal of Medicinal Chemistry.– 2003.– № 46.– C. 499.

32 Jones, G. Development and Validation of a Genetic Algorithm for Flexible Docking / G. Jones //Journal of Molecular Biology.– 1997.– № 267.– C. 727.

33 Najmanovitch, R. Side-chain flexibility in proteins upon ligand binding / R. Najmanovitch //Proteins.– 2000.– № 39.– С. 261.

34 Mangoni, M. Docking of flexible ligands to flexible receptors in solution by molecular dynamics simulation / M. Mangoni, D. Roccatano. A. Di Nola //Proteins.– 1999.– № 35.– С. 153.

35 Davis, I.W. RosettaLigand docking with full ligand and receptor flexibility / I.W. Davis, D. Baker // Journal of molecular biology.– 2009.– Т. 385, №2.– С.381.

36 Meiler, J. ROSETTALIGAND: Protein–Small Molecule Docking with Full Side-Chain Flexibility / J. Meiler, D. Baker // PROTEINS: Structure, Function, and Bioinformatics.– 2006.– № 65.– С. 538.

37 Katchalski-Katzir, E. Molecular surface recognition—determination of geometric fit between proteins and their ligands by correlation techniques / E. Katchalski-Katzir // Proceedings of the National Academy of Sciences.– 1992.– № 89.– С. 2195.

38 Gabb, H.Modeling protein docking using shape complementarity, electrostatics, and biochemical information / H. Gabb, R. Jackson, M. Sternberg // Journal of Molecular Biology.– 1997.– № 272.– С. 106.

39 Morris, G.M. Automated docking using a Lamarckian genetic algorithm and an empirical binding free energy function / G.M. Morris [идр.] // Journal of Computational Chemistry.– 1998.– Т.19.– С. 1639.

40 Романов, А.Н. Компьютерная разработка лекарств: программа докинга SOL / А.Н. Романов [и др.] // Вычислительные методы и программирование.– 2008.– Т.9, №2.– С. 64.

41 Офёркин, И.В. Реализация поддержки параллельных вычислений в программах докинга SOLGRID и SOL/ И.В. Офёркин// Вычислительные методы и программирование.– 2011.– Т. 12, № 1.– С. 205.

42 Stroganov, O.V. Lead finder: an approach to improve accuracy of protein-ligand docking, binding energy estimation, and virtual screening / O.V. Stroganov[идр.]// Journal of Chemical Information and Modeling.– 2008.– Т. 48, № 12.– С. 2371.

43 Blattner, T. Improving Protein Docking Efficiency with General Purpose Computation for Graphics Processing Units - Formal Report [Электронныйресурс] / T. Blattner [идр.].–Режим доступа: http://gpuautodock.sourceforge.net.

44 Kannan, S. Porting Autodock to CUDA / S. Kannan, R. Ganji // In proceeding of Evolutionary Computation.– 20010.– С. 1.

45 Abagyan, R. Hybrid Metaheuristics. An Emerging Approach to Optimization / R. Abagyan, M. Totrov, D. Kuznetsov // Journal of Computational Chemistry.– 1994.– № 15.– С. 488.

46 Nocedal, J. Numerical Optimization / J. Nocedal, S. Wright.–Изд-воSpringer, 1999.– 634 с.

47 DesJarlais, R.L. Docking flexible ligands to macromolecular receptors by molecular shape / R.L. DesJarlais [идр.] // Journal of Medicinal Chemistry.– 1986.– № 29.– С. 2149.

48 Rarey, M. Time-Efficient Docking of Flexible Ligands into Active Sites of Proteins / M. Rarey, B.Kramer, T.Lengauer // In proceedings of ISMB-95.– 1995.– 300 с.

49 Apostolakis, J. Docking Small Ligands in Flexible Binding Sites / J. Aposrolakis, A. Pluckthun, A. Caflisch // Journal of Computational Chemistry.– 1998.– Т. 19 № 1.– С. 21.

50 Totrov, M. Flexible protein-ligand docking by global energy optimization in internal coordinates / M. Totrov, R. Abagyan // Protein.– 1997.– № 1(Suppl).– С. 215.

51 Korb, O. PLANTS: Application of Ant Colony Optimization to Structure-Based Drug Design / O. Korb, T. Stützle, T.E. Exner // In Ant Colony Optimization and Swarm Intelligence, 5th International Workshop, ANTS 2006.– 2006.–Т. 4150.– С. 247.

52 Korb, O. Accelerating molecular docking calculations using graphics processing units / O. Korb, T. Stützle, T.E. Exner // Journal of chemical information and modeling. American Chemical Society.– 2011.– Т. 51№ 4.– С. 865.

53 Brooijmans, N. Molecular recognition and docking algorithms / N. Brooijimans, I. Kuntz //Annual Review of Biophysics and Biomolecular Structure.– 2003.– № 32.– С. 335.

54 Cornell, W.D. A Second Generation Force Field for the Simulation of Proteins / W.D. Cornel [идр.] // Journal of the American Chemical Society.– 1995.– Т. 117№ 19.– С. 5179.

55 MacKerell, A.D. All-Atom Empirical Potential for Molecular Modeling and Dynamics Studies of Proteins / A.D. MacKerel [идр.] // The Journal of Physical Chemistry B.– 1998.– Т. 102№ 18.– С. 3586.

56 MacKerell, A.D. Atomistic Models and Force Fields //A.D. MacKerel [идр.] / Computational Biochemistry and Biophysics.– 2001.– С. 7.

57 Meng, E. Automated docking with grid-based energy evaluation / E. Meng, B. Shoichet, I. Kuntz // Journal of Computational Chemistry.– 1992.– Т. 13№ 4.– С. 505.

58 Weiner, S.J. An all atom force field for simulations of proteins and nucleic acids / S.J. Weiner [идр.]// Journal of Computational Chemistry.– 1986.– Т. 7№ 2.– С. 230.

59 Weiner, S.J. A New Force Field for Molecular Mechanical Simulation of Nucleic Acids and Proteins / S.J. Weiner [идр.] // Journal of the American Chemical Society.– 1984.– Т. 106№ 3.– С. 765.

60 Jorgensen, W.L. Development and testing of the OPLS all-atom force field on conformational energetics and properties of organic liquids / W.L. Jorgensen, , D.S. Maxwell, J. Tirado-Rives //Journal of the American Chemical Society.– 1996.– Т. 118№ 45.– С. 11225.

61 Gilson, M.K. Calculation of Protein-Ligand Binding Affinities / M.K. Gilson, H. Zhou //Annual Review of Biophysics and Biomolecular Structure.– 2007.– № 36.– С. 21.

62 Weiner, P. AMBER: Assisted model building with energy refinement. A general program for modeling molecules and their interactions / P. Weiner, P. Kollman // Journal of Computational Chemistry.– 1981.– Т. 2 № 3.– С. 287.

63 Brooks, B.R.CHARMM: A program for macromolecular energy, minimization, and dynamics calculations / B.R. Brooks [идр.] // Journal of Computational Chemistry.– 1983.– Т. 2 № 2.– С. 187.

64 Halgren, T. Merck molecular force field. I. Basis, form, scope, parameterization, and performance of MMFF94 / T. Halgren // Journal of Computational Chemistry.– 1996.– Т. 17№ 5-6.– С. 490.

65 Allinger, N. Conformational analysis. 130. MM2. A hydrocarbon force field utilizing V1 and V2 torsional terms // N. Allinger //Journal of the American Chemical Society.– 1977.– Т. 99№ 25.– С. 8127.

66 Allinger, N. Molecular Mechanics. The MM3 Force Field for Hydrocarbons. / N. Allinger,Y.H. Yuh, J. Huei // Journal of the American Chemical Society.– 1989.– Т. 111№ 23.– С. 8551.

67 Huei, J.Molecular mechanics. The MM3 force field for hydrocarbons. 2. Vibrational frequencies and thermodynamics / J. Huei, N. Allinger // Journal of the American Chemical Society.– 1989.– Т. 111№ 23.– С. 8566.

68 Фарков, М.А. Специфика предобработки биоинформационных данных для графических процессоров / М.А. Фарков // Сборник трудов всероссийской научно-практической конференции «Многоядерные процессоры, параллельное программирование, ПЛИС, системы обработки сигналов».– Барнаул.– 2013.– С.76.

69 Guerrero, G. Effective Parallelization of Non-bonded Interactions Kernel for Virtual Screening on GPUs / G. Guerrero [идр.] // In proceeding of 5th International Conference on Practical Applications of Computational Biology & Bioinformatics (PACBB 2011).– 2011.– С. 63.

70 Roh, Y. A molecular docking system using CUDA / Y. Roh [идр.] // In Proceedings of the 2009 International Conference on Hybrid Information Technology (ICHIT '09).– 2009.–С. 28.

71 Wang, J. Automatic atom type and bond type perception in molecular mechanical calculations / J. Wang [идр.] //Journal of molecular graphics modelling.– 2006.– Т. 25№ 2.– С. 247.

72 Wang, J.Development and testing of a general AMBER force field / J. Wang //Journal of Computational Chemistry.– 2004.– № 25.– С. 1157.

73 Фарков, М.А. Сеточныйподходквыполнениюмолекулярногодокинга. Егодостоинстваинедостатки. / М.А. Фарков// Сборник трудов X Международной

научно-практической конференции студентов, аспирантов и молодых ученых «Молодежь и современные информационные технологии».– Томск.– 2012.– С. 186.

74 Фарков, М.А. Ускорение вычисления сеток силовых полей при помощи графических процессоров / М.А. Фарков // Сборник трудов XIII Международной молодёжной научной конференции «Интеллект и наука труды».– Железногорск.– 2013.– С. 129.

75 Фарков, М.А. Вычисление сеток взаимодействия молекул с использованием графических процессоров / М.А. Фарков // Исследования наукограда.– 2014.–Т. 21, № 3.–С. 49.

76 Farkov, M. CalculationofforcefieldgridsformoleculardockingusingGPU / M. Farkov // JournalofSiberianFederalUniversity. Biology.–2014.–Т.7, № 1.– С. 4.

77 Storn, R. Differential evolution - A simple and efficient heuristic for global optimization over continuous spaces / R. Storn, K. Price // Journal of Global Optimization.–1997.–Т. 11, № 4.– С. 341.

78 de Veronese, L.P. Differential evolution algorithm on the GPU with C-CUDA / L.P. de Veronese, R.A. Krohling // Evolutionary Computation (CEC).– 2010.– С. 1.

79 Gonzalez, D. Fully parallel differential evolution / D. Gonzalez, N.G. Barriga // In Competition of GPUs for Genetic and Evolutionary Computation at the 2011 Genetic and Evolutionary Computation Conference. GECCO’2011.– 2011.– С. 1.

80 Kromer, P. A comparison of many-threaded differential evolution and genetic algorithms on CUDA / P.A. Kromer [идр.] // Nature and Biologically Inspired Computing.–2011.– С. 509.

81 Ramirez-Chavez, L.E. A GPU-Based Implementation of Differential Evolution for Solving the Gene Regulatory Network Model Inference Problem / L.E. Coello, C.A. Coello, E. Rodriguez-Tello // In proceeding of: Proceedings of the Fourth International Workshop on Parallel Architectures and Bioinspired Algorithms, WPABA 2011.–2011.– С. 1.

82 Davendra, D. GPU based enhanced differential evolution algorithm: a computational analysis / D. Davendra [идр.] // In Proceedings of 26th European Conference on Modelling and Simulation.– 2012.– С. 1.

83 Qin, A.K. An improved CUDA-based implementation of differential evolution on GPU / A.K. Qin [и др.]// In Proceedings of the 14th annual conference on Genetic and evolutionary computation. GECCO '12.– 2012.–С.991.

84 Фарков, М.А. Улучшениереализациииметодадифференциальнойэволюциинаграфическихпроцессорах / М.А. Фарков, А.ИЛегалов // Вестник Сибирского государственного аэрокосмического университета имени академика М.Ф. Решетнева.– 2014.–Т. 3, № 55.–С. 157.

85 Фарков, М.А. Применение методов оптимизации для выполнения молекулярного докинга на графических процессорах /М.А. Фарков, А.И. Легалов // Моделирование и анализ информационных систем.– 2014.–Т. 21, № 5.–С. 93.

86 NVIDIA CUDA C Programming Guide [Электронныйресурс].– NVIDIA CUDA C Programming Guide.– Режим доступа http://docs.nvidia.com/cuda/cuda-c-programming-guide.

87 Kirk, D. Programming Massively Parallel Processors: A Hands-on Approach / D. Kirk, H.W Hwu.– 2010.– 280 с.

88 Sanders, J. CUDA by Example / J. Sanders, E. Kandrot.–Изд-во: Addison-Wesley.– 2010.– 279с.

89 Wilt, N. The CUDA Handbook A Comprehensive Guide to GPU Programming / N. Wilt.–Изд-во: Addison-Wesley.– 2013.– 494с.

90 Nickolls, J. Scalable parallel programming with CUDA / J. Nickolls [идр.] // Queue.–2008.–Т. 6, № 2.– С. 40.

91 Garland, M. Parallel computing experiences with CUDA / M. Garland // Proceeding IEEE Micro.–2008.–Т . 28, № 4.–С. 13.

92 Боресков, А. ОсновыработыстехнологиейCUDA/А. Боресков, А. Харламов.–Изд-во: ДМКПресс.– 2012.– 232с.

93 Cook, S. CUDA Programming / S. Cook.–Изд-во: Elsevier.– 2013.–С. 576.

94 McCool, M.D. Scalable programming models for massively multicore processors / M.D. McCool // ProceedingIEEE.– 2008.– Т. 96, № 5.–С. 816.

95 Garland, M. Parallel computing with CUDA / M. Garland // Proceeding IPDPS 2010.–2010.

96 Buck, I. GPU Computing: Programming a Massively Parallel Processor / I. Buck // Proceeding of International Symposium on Code Generation and Optimization.– 2007.

97 Luebke, D. CUDA: Scalable parallel programming for high-performance scientific computing / D. Luebke // Proceeding 2008 5th IEEE International Symposium on Biomedical Imaging: From Nano to Macro, Proceedings, ISBI.– 2008.–С. 836.

98 Yang, C.T. Hybrid CUDA, OpenMP, and MPI parallel programming on multicore GPU clusters / C.T. Yang, C.L. Huang, C.F. Lin // Computer Physics Communications.–2011.–Т. 182, № 1.–С. 266.

99 Yang, C.T. Hybrid parallel programming on GPU clusters / C.T. Yang [идр.] // Proceedings - International Symposium on Parallel and Distributed Processing with Applications, ISPA 2010.–2010.–С. 142.

100 Che, S. A performance study of general-purpose applications on graphics processors using CUDA / S. Che[идр.] // Journal of Parallel and Distributed Computing.– 2008.–Т. 68, № 10.–С. 1370.

101 Karunadasa, N.P. Accelerating high performance applications with CUDA and MPI / N.P Karunadas, D.N. Ranasinghe // Proceeding ICIIS 2009 - 4th International Conference on Industrial and Information Systems.– 2009.– С. 331.

102 Lindholm, E. NVIDIA Tesla: A unified graphics and computing architecture / E. Lindholm [идр.]// Proceeding IEEE Micro.– 2008.–Т. 28, № 2.– С. 39.

103 Ghorpade, J. GPGPU Processing in CUDA Architecture / J. Ghorpade // Advanced Computing: An International Journal.– 2012.–Т. 3, № 1.– С. 105.

104 Munshi, A. The OpenCL Specification / A. Munshi // Khronos OpenCL Working Group.–2011.– 385 с.

105 Tsuchiyama, R. The OpenCL Programming Book / R. Tsuchiyama.– Изд-во: GroupFixstars Corporation.– 2010.– 245 с.

106 Stone, J.E.OpenCL: A parallel programming standard for heterogeneous computing systems / J.E. Stone, D. Gohara, G. Shi // Computing in Science & Engineering.– 2010.–Т. 12, № 3.– С. 66.

107 Gaster, B. Heterogeneous Computing with OpenCL / B. Gaster [идр.] // Heterogeneous Computing with OpenCL.– 2012.–С. 67.

108 Fang, J. A comprehensive performance comparison of CUDA and OpenCL / J. Fang, A.L. Varbanescu, H. Sips. // Proceedings of the International Conference on Parallel Processing.–2011.–С. 216.

109 Jeffers, J. Intel Xeon Phi Coprocessor High Performance Programming / J. Jeffers, J. Reinders // Vasa.– 2013.–С. 409.

110 Simmler, H. Multitasking on FPGA Coprocessors / H. Simmler, L. Levinson // Field-Programmable Logic and Applications. ACM.– 2000.–С. 121.

111 Farooq, U.FPGA Architectures: An Overview / U. Farooq, Z. Marrakchi, H. Mehrez // Tree-based Heterogeneous FPGA Architectures.– 2012.–С. 7.

112 Kuon, I.FPGA Architecture: Survey and Challenges / I. Kuon, R. Tessier, J. Rose // Foundations and Trends® in Electronic Design Automation.– 2007.–Т. 2, № 2.– С. 135.

113 Stützle, T. MAX-MIN Ant System / T. Stützle, H.H. Hoos // Future Generation Computer Systems.– 2000.–Т. 16, № 8.– С. 889.

114 Griewank, A.O. Generalized descent for global optimization / A.O. Griewank // Journal of Optimization Theory and Applications.– 1981.–Т. 34, № 1.– С. 11.

115 Molga, M. Test functions for optimization needs [Электронныйресурс] / M. Molga, C. Smutnicki.– Режим доступа: http://www.robertmarks.org/Classes/ENGR5358/Papers/functions.pdf.

116 Nelder, J.A. A Simplex Method for Function Minimization / J.A. Nelder, R. Mead // Computer Journal.– 1965.–Т. 7, № 4.– С. 308.

117 Rosenbrock, H.H. Some general implicit processes for the numerical solution of differential equations / H.H. Rosenbrock // ComputerJournal.– 1963.–Т. 5, № 4.– С. 329.

118 Hooke, R. Direct Search Solution of Numerical and Statistical Problems / R. Hooke, T.A. Jeeves // Journal of the ACM.– 1961.–Т. 8, № 2.– С. 212.

119 Hestenes, M.R. Methods of Conjugate Gradients for Solving Linear Systems / M.R. Hestenes, E. Stiefel //Journal of Research of the National Bureau of Standards.– 1934.–Т. 49, № 6.– С. 409.

120 Dai, Y.H. A Nonlinear Conjugate Gradient Method with a Strong Global Convergence Property / Y.H. Dai, Y. Yuan // SIAM Journal of Optimization.– 1999.–Т. 10, № 1.– С. 177.

121 Yuan, Y. Step-sizes for the gradient method // AMS/IP Studies in Advanced Mathematics.– 2008.– С. 785.

122 Пантелеев,А.В. Методыоптимизациивпримерах и задачах: Учеб. Пособие/А.В. Пантелеев, Т.А. Летова. - 2-е изд., исправл. - М.: Высш. шк., 2005.– 544 с.

123 Metropolis, N. The Monte Carlo method / N. Metropolis, S. Ulam // Journal of the American Statistical Association.– 1949.–Т. 44, № 247.– С. 335.

124 Whitley, D. A genetic algorithm tutorial / D. Whitley // Statistics and Computing.– 1994.–Т. 4, № 2.– С. 65.

125 Dorigo, M. Ant system: Optimization by a colony of cooperating agents / M. Dorigo, V. Maniezzo, A. Colorni // IEEE Transactions on Systems, Man, and Cybernetics, Part B.– 1996.–Т. 26, № 1.– С. 29.

126 Kennedy, J. Particle swarm optimization / J, Kennedy, R. Eberhart // Proceeding of IEEE International Conference.– 1995.–Т. 4.– С. 1942.

127 Shi, Y. A modified particle swarm optimizer / Y. Shi, R. Eberhart // Proceeding of IEEEEvolutionary Computation Proceedings.–1998.

128 Björck, M. GenX: An extensible X-ray reflectivity refinement program utilizing differential evolution / M. Björck, G. Andersson // Journal of Applied Crystallography.– 2007.–Т. 40, №6.– С. 1174.

129 Seaton, C.C. Differential evolution: crystal structure determination of a triclinic polymorph of adipamide from powder diffraction data / C.C. Seaton, M. Tremayne // Chemical Communications.– 2002.–№8.– С.880.

130 Mendes, S.P.A Differential Evolution Based Algorithm to Optimize the Radio Network Design Problem / S.P. Mendes [идр.] // Proceeding Second IEEE Int. Conf. e-Science Grid Comput.– 2006.

131 Coelho, L. Differential evolution optimization combined with chaotic sequences for image contrast enhancement / L. Coelcho [идр.] // Chaos, Solitons and Fractals.– 2009.–Т. 42, №1.– С.522.

132 Qing, A.Q.A. Dynamic differential evolution strategy and applications in electromagnetic inverse scattering problems / A.Q.A. Quing // Proceeding IEEE Transactions on Geoscience and Remote Sensing.– 2006.–Т. 44, №1.

133 Price, K.Differential Evolution: A Practical Approach to Global Optimization (Natural Computing Series) / K. Price, R.M. Storn, J.A. Lampinen // Journal of Heredity.– 2005.–Т. 104.– С.542.

134 Costa, W.T. Identification of photovoltaic model parameters by differential evolution / W.T. Costa [идр.] // Proceedings of the IEEE International Conference on Industrial Technology.– 2010.–С. 931.

135 Nomenclature A. Nomenclature and symbolism for amino acids and peptides (Recommendations 1983) // Pure and Applied Chemistry.– 1984.– Т. 56, № 5.– С. 595.

Приложение А. Акты о внедрении

Для представления
в диссертационный совет

Акт о внедрении

Настоящим удостоверяется, что рекомендации, содержащиеся в диссертации Фаркова Михаила Александровича на соискание учёной степени кандидата технических наук, были использованы в ООО «Мобилфон» для решения задач оптимизации распределения вычислений в многосерверной вычислительной среде.

Генеральный директор Камоцкий А.С.

660124, г. Красноярск, ул. Борисевича 8 пом. 4
тел: 8 (391) 2 206 401; факс: 8 (391) 2 206 404
e-mail: info@mobilfon.ru, www.mobilfon.ru

Borisevicha st. 8 office 4, Krasnoyarsk, Russia, 660124
tel.: 8 (391) 2 206 401; Fax: 8 (391) 2 206 404
e-mail: info@mobilfon.ru, www.mobilfon.ru

Общество с ограниченной ответственностью «Функциональные наносисистемы»
662974, г.Красноярск, ул.Академика Киренского д.26ж, оф.302
ИНН / КПП 2463231859/246301001
р/с 40702810031280030817 в в Восточно-Сибирском банке Сбербанка России
к/с 30101810800000000627 БИК 040407627
Тел/факс (391) 290-55-79 email: Funnano@mail.ru

Для представления в
диссертационный совет

Акт о внедрении

Настоящим удостоверяется, что рекомендации, содержащиеся в диссертации Фаркова Михаила Александровича на соискание учёной степени кандидата технических наук, были использованы в ООО «ФанНано» в рамках решения задачи расчёта кинетики реакции и транспорта на границе раздела "металлический микросетчатый электрод - органическая электрохромная композиция".

Директор Хартов С.В.