Министерство Образования и Науки Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего профессионального образования
Московский Физико-Технический Институт
(Государственный Университет)

Кафедра вычислительной математики

Молекулярное моделирование с использованием графических процессоров

Жмуров А.А., Барсегов В.А.

Учебное пособие

Москва - 2013

Содержание

Введение в вычислительную биологию.

Вычислительный эксперимент как новый инструмент для научных исследований

Вычислительный эксперимент берёт своё начало со времён развития первых компьютеров в 40-х – 50-х годах прошлого века, а первое применение компьютеров для молекулярной динамики датируется 1956 годом. В наше время вычислительный эксперимент стал стандартным инструментом, используемым исследователями для большого спектра задач которые не поддаются аналитическому решению (с помощью "карандаша и бумаги").

Вычислительный эксперимент тесно связан как с натурным экспериментом, так и с теоретическими исследованиями: Эксперимент $\leftrightarrow$ Моделирование $\leftrightarrow$ Теория.

1. Модели, используемые в вычислительном эксперименте, базируются на теоретическом приближении. Соответственно, вычислительный эксперимент может быть использован для проверки теоритических гипотез.

2. Вычислительный эксперимент сам по себе не предлагает понимания физических основ исследуемого процесса. Результаты вычислений необходимо обрабатывать при помощи теоретических знаний.

3. Вычислительный эксперимент не может полностью заменить эксперимент натурный. При использовании вычислительного эксперимента результат зависит от изначально заданной модели, поэтому вычисления очень тесно связано с экспериментом и требует экспериментальных данных для проверки результата. Часто натурный эксперимент предоставляет начальные условия для вычислительного эксперимента.

Вычислительный эксперимент может быть использован для предсказания свойств материалов в экстремальных условиях (высокой температуре, давлении) или для изучения свойств сложных систем (например, сворачивание белков или их агрегация). Последнее

особенно важно, так как благодаря этим возможностям вычислительного эксперимента, он сейчас находится на острие научных исследований.

Развитие вычислительной биологии

Вычислительная биология включает в себя практически любое применение вычислительных технологий при изучении биологических систем. Комбинация растущего количества экспериментальных данных и увеличивающейся производительности компьютеров является фундаментом развития вычислительной биологии. К задачам вычислительной биологии можно отнести:

1. Сравнение и анализ генетических последовательностей (биоинформатика).
2. Изучение зависимости между генетическими последовательностями, структурами биомолекул и их функциями (функциональная геномика).
3. Предсказание биомолекулярных структур при помощи вычислительных технологий (структурная геномика).
4. Изучение биофизических свойств биомолекул (биомолекулярное моделирование).

Последние два пункта являются первичной целью вычислительных экспериментов в биологии, использование котор позволяет:

1. Изучать динамику структурных изменений в биомолекулах, включая агрегацию или диссоциацию белок-белковых комплексов, образование нативных состояний (фолдинг) и денатурацию белков.
2. Изучать термодинамические свойства биомолекулярных систем.
3. Предсказывать эффект изменения внешних условий, таких, как температура, кислотность среды, наличие денатурирующего вещества.
4. Предсказывать нативную структуру из последовательности аминокислот.

От последовательности к структуре

Определение последовательности белков, ДНК и РНК может быть произведено при помощи сравнительно простых экспериментальных технологий. Благодаря этому, научное сообщество уже определило последовательности для очень большого числа биомолекул. Однако, функции биологических молекул тесно связаны с их нативным состоянием, определить которое экспериментально может быть проблематичным. Во-первых, используемые экспериментальные методики имеют ряд ограничений и не лишены погрешностей. Рентгеновская кристаллография, например, требует наличия периодического кристалла, в ячейках которого находятся копии исследуемой молекулы. Силы взаимодействия между ячейками кристалла могут повредить исследуемую молекулу, а некоторые белки вообще не поддаются кристаллизации. Во-вторых, проведение натурного эксперимента требует существенных временных и материальных затрат. На сегодняшний день, количество разрешённых последовательностей существенно (примерно в 100 раз) превосходит количество известных трёхмерных структур [1]. Использование вычислительного эксперимента может позволить сократить этот разрыв. Уже известны примеры успешного применения вычислительного эксперимента для разрешения белковых структур, но, в связи с колоссальными вычислительными затратами, такие примеры пока единичны [2]. Несмотря на это, в связи с постоянным ростом вычислительных возможностей и появлением вычислительных устройств нового типа, применение вычислительного эксперимента в этой области выглядит многообещающим.

Структура пособия

Цель данного пособия – обеспечить понимание основных принципов молекулярного моделирования с использованием нескольких уровней детализации. В первой части пособия дан обзор строения различных биологических молекул – от последовательности до трёхмерной структуры. Далее разъяснены предпосылки и методы молекулярной механики и динамики, а также рассмотрены предпосылки использования различных подходов к моде-

лированию. В качестве первого практического примера рассмотрена простая задача моделирования с применением общедоступного бесплатного пакета NAMD, разъяснены основы метода молекулярной динамики и использования программного пакета VMD для визуализации и анализа результатов. В четвёртой части пособия приведены основы устройства графических процессоров, а также пакета CUDA, используемого для написания программ, работающих на этих высокопроизводительных устройствах. В последней части шаг за шагом рассмотрен ещё один пример, в котором читателю предлагается с нуля написать собственную программу для молекулярного моделирования на ГП. На протяжении всего курса разъясняется формат основных файлов, используемых в моделировании.

Часть I

Биологические молекулы

1 Вода

Вода – один из составляющих элементов биологических систем. Человеческий организм на 70% состоит из воды. Для того, чтобы понять её важность для нормальной функции живого организма, необходимо сначала остановиться на структуре молекулы воды. Молекула воды состоит из двух атомов водорода и одного атома кислорода и имеет массу в 18 а.е.м. (атомных единиц массы). Структура электронных облаков входящего в состав воды кислорода напоминает форму тетраэдра, две вершины которого заняты атомами водорода, а две – свободными электронами. Атом кислорода "перетягивает" отрицательный заряд, и две ковалентные связи $O-H$ в молекуле воды образуют диполи с частичным положительным зарядом на атоме водорода и компенсирующим отрицательным зарядом на атоме кислорода. Равновесное значение угла между двумя ковалентными связями $O-H$ составляет примерно 104,5°, а равновесная длина – примерно 0,958Å (Рис. 1).

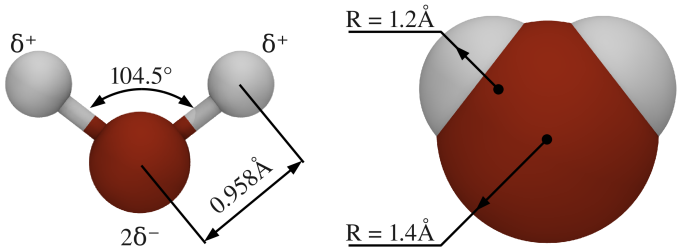


Рис. 1: Структура молекулы воды. Два атома водорода валентно связаны с атомом кислорода. Ковалентные связи длиной в $\sim 0,958$Å образуют угол в 104,5°. Справа показаны сферы Ван-дер-Ваальса для атомов водорода и кислорода.

Молекулы воды обладают как частичными положительным так и отрицательным зарядами, что заставляет соседние молекулы притягиваться друг к другу. Подобные связи носят электростатическую природу и обычно называются водородными. Последние образуются

между атомом водорода, валентно связанным с атомом кислорода и частично заряженным атомом кислорода соседствующей молекулы воды. В результате, атом водорода делится между двумя атомами кислорода, с одним из которых он связан валентно (обозначается $O{-}H\ldots O$). Так как молекула воды состоит только из атома кислорода и двух атомов водородов, количество водородных связей в воде колоссально.

Средняя энергия водородной связи – 20КДж/моль, что существенно меньше энергии валентной связи $O{-}H$, составляющей примерно 460КДж/моль. Несмотря на то, что единичная водородная связь слаба, их большое количество определяет плотность воды и температуры её фазовых переходов. В жидкой фазе каждая молекула воды образует в среднем три из четырёх возможных водородных связей (Рис. 2). Это происходит из-за того, что молекулы воды находятся в постоянном движении, взаимное положение молекул воды постоянно меняется, водородные связи разрушаются и образуются вновь: время жизни одной водородной связи в воде – порядка 10^{-12} секунды. В твёрдой фазе количество водородных связей на одну молекулу воды максимально и равно четырём. В силу особенностей геометрической структуры молекулы воды, такая компоновка требует большего пространства и вода в твёрдой фазе (лёд) обладает меньшей плотностью, чем вода в жидкой фазе.

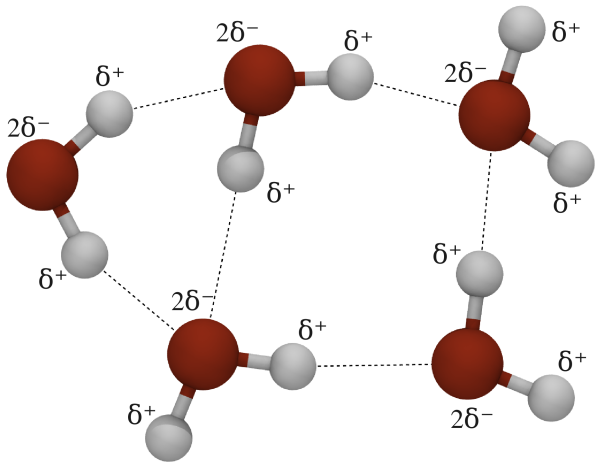


Рис. 2: Водородные связи между молекулами воды в жидкой фазе (показаны пунктиром).

Полярная структура воды позволяет ей хорошо растворять заряженные молекулы (ио-

ны). Положительно заряженные атомы водорода ориентируются в сторону отрицательно заряженных частей растворённого вещества, а отрицательно заряженные атомы кислорода – в сторону положительно заряженных частей (Рис. 3). Говорят, что молекулы воды формируют “шубу” вокруг растворённых веществ. Незаряженные и неполярные вещества в воде не растворяются: молекулам воды энергетически более выгодно находиться в постоянном контакте друг с другом. Любое незаряженное неполярное (не имеющее диполей) вещество (например, масло), помещённое в воду, будет собираться в большие капли, которые позволяют сделать поверхность соприкосновения с водой минимальной. Поэтому незаряженные неполярные вещества называются гидрофобными (“боящимися воды”). То, что гидрофобные молекулы “склеиваются”, будучи помещёнными в воду, называют гидрофобным эффектом. Этот эффект очень важен в процессе образования и стабилизации нативной структуры биологических молекул. Схожий эффект также объясняют возникновение сил поверхностного натяжения.

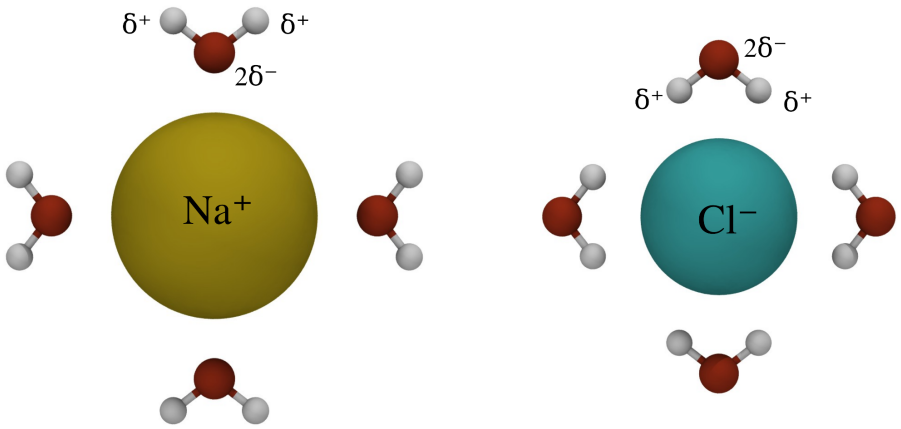


Рис. 3: Ориентация молекул воды вокруг атомов положительно (слева) и отрицательно (справа) заряженного ионного раствора.

2 Белки

Множество биологических функций организма осуществляется белками. Эти функции включают в себя перенос и хранение кислорода (гемоглобин и миоглобин соответственно), сокращение мускул (титин), рост костей (коллаген), слияние клеток (фибринонектин) и так далее. Разнообразие в функциях белков предполагает разнообразие в их структуре. Действительно, белки могут быть очень разными как по форме, так и по размеру. Один из самых больших белков – титин – содержит порядка 30 000 аминокислот, сгруппированных в приблизительно 300 доменов. Коллаген содержит три независимые цепи, свёрнутые в тройную спираль. Один из самых маленьких белков, домен WW, содержит от 35 до 40 аминокислот. Большие белки часто формируют отдельные домены, количество аминокислот в каждом из которых редко превышает 100. Далее строение белков и их разнообразие будет рассмотрено более детально – от атомарной структуры до их нативного состояния.

2.1 Аминокислоты

Белки — высокомолекулярные органические вещества, состоящие из соединённых пептидной связью в цепочку аминокислот. В живых организмах аминокислотный состав белков определяется генетическим кодом. При синтезе в большинстве случаев используется 20 стандартных аминокислот, которые отличаются друг от друга химической формулой свободного радикала. Множество комбинаций аминокислот определяет большое разнообразие свойств молекул белков. Часто в живых организмах несколько молекул белков образуют сложные комплексы, в которых каждому белку отведена отдельная функция.

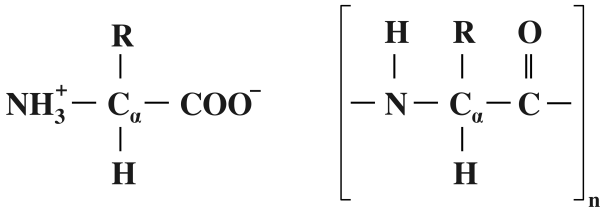


Рис. 4: Структура аминокислоты и полипептида.

Последовательность аминокислот определяет первичную структуру белка. Каждая аминокислота содержит центральный C_α-углерод, водород, группы NH_3^+ и COO^- и боковой радикал R (Рис. 4). Такая химическая формула аминокислоты типична при нормальной кислотности среды (pH). В процессе синтеза белков рибосомой используется 20 различных аминокислот. Общая структура аминокислоты внутри белка показана на Рис. 4 справа. По своим свойствам аминокислоты могут быть примерно разделены на гидрофобные, полярные (или гидрофильные) и заряженные. Существует и другая классификация аминокислот — по химическому составу бокового радикала. Боковой радикал алифатических аминокислот состит из атомов углерода и водорода. В некоторых аминокислотах к алифатической цепи ковелентно присоеденена карбоксильная группа OH. Также существуют аминокислоты с щелочным, ксилотным и ароматическими боковыми радикалами. Далее рассмотрим свойства и атомарное строение всех 20-ти аминокислот.

2.1.1 Алифатические аминокислоты

К этому типу аминокислот относятся глицин, аланин, валин, люсин и изолюсин. Алифатические аминокислоты гидрофобны (структуры показаны на рисунке 5, где они упорядочены по увеличению гидрофобности). Они обладают открытым, иногда разветвляющимся радикалом. Алифатические аминокислоты не формируют водородных связей, они нейтральны (не заряжены) и неполярны. Самой маленькой аминокислотой является глицин, у которого отсутствует радикал, и который обладает большой гибкостью пептидной связи. С ростом радикала гибкость пептидной связи уменьшается.

2.1.2 Алифатические аминокислоты с гидроксидной группой

К этому типу аминокислот относятся серин и треонин, которые содержат группу OH в радикале (Рис. 6). Эти аминокислоты полярны, способны образовывать водородные связи и гидрофильны. Боковые радикалы этих аминокислот нейтральны (не заряжены).

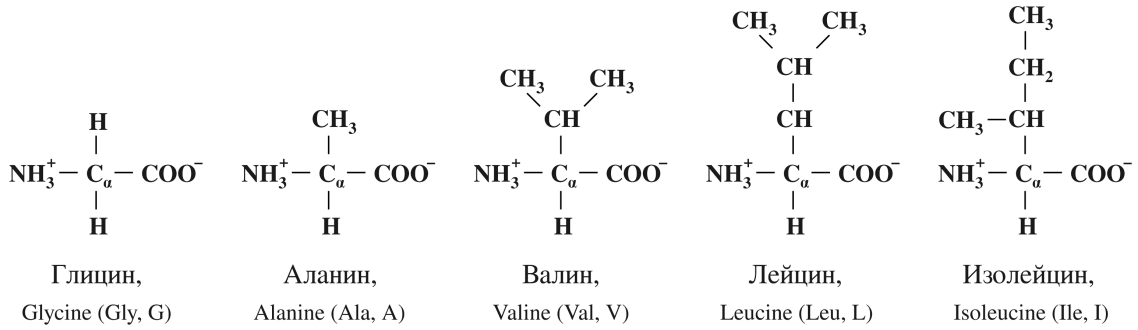


Рис. 5: Алифатические аминокислоты.

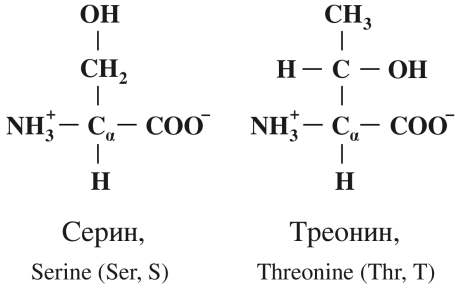


Рис. 6: Алифатические аминокислоты с гидроксидной группой.

2.1.3 Кислотные аминокислоты и их производные

Эта группа включает в себя аспарагин, глутамин, аспарагиновую кислоту и глутаминовую кислоту (Рис. 7). Аспарагин и глутамин содержат амидную группу (NH) на своих боковых радикалах и являются незаряженными аминокислотами. Аспарагиновая и глутаминовая кислоты содержат депротонированую гидроксильную группу (COO^-) и несут на себе отрицательный заряд при кислотности среды выше pH$\sim$4. Все четыре кислотных аминокислоты полярны (и гидрофильными) и способны формировать водородные связи.

2.1.4 Щелочные аминокислоты

Аминокислоты этой группы – лизин, аргинин и гистидин (Рис. 8). Лизин и аргинин содержат щелочную группу, которая положительно заряжена (протонирована) при нормальной кислотности среды. Значение pK (величина pH, при которой их боковые радикалы

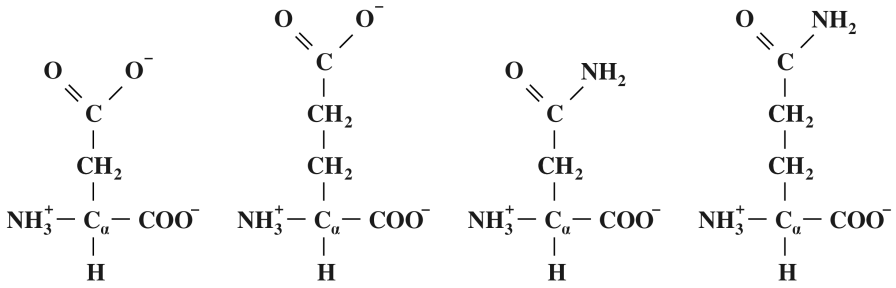


Рис. 7: Кислотные аминокислоты и их производные.

становятся нейтральными) находится в диапазоне от 10 до 12. Боковой радикал гистидина имеет значение pK=6.5, поэтому его заряд очень чувствителен к изменениям кислотности окружающей среды. Щелочные аминокислоты сильно полярны и способны образовывать водородные связи.

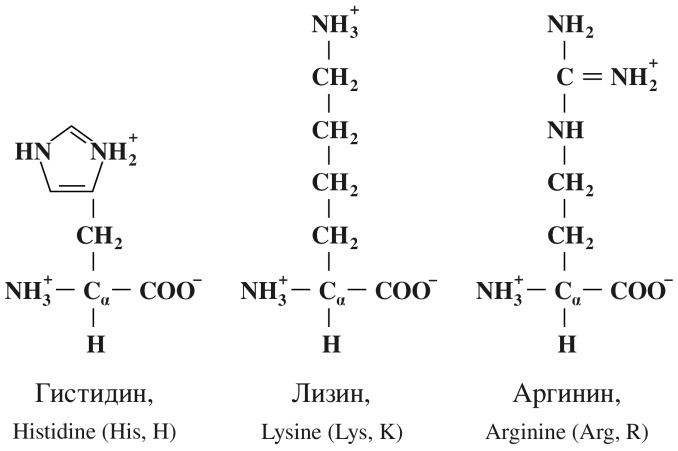


Рис. 8: Щелочные аминокислоты.

2.1.5 Ароматические аминокислоты

Боковые радикалы аминокислот фениаланин, тирозин, триптофан содержат циклические группы, из-за этого они главным образом гидрофобны, хотя степень гидрофобности

может меняться (Рис. 9). Фениаланин – самая гидрофобная из трёх, а тирозин и триптофан обладают некоторой полярностью боковых радикалов, что делает их менее гидрофобными. Эти аминокислоты нейтральны в нормальных условиях. Тирозин и триптофан могут образовывать водородные связи, в то время как фенилаланин – не может.

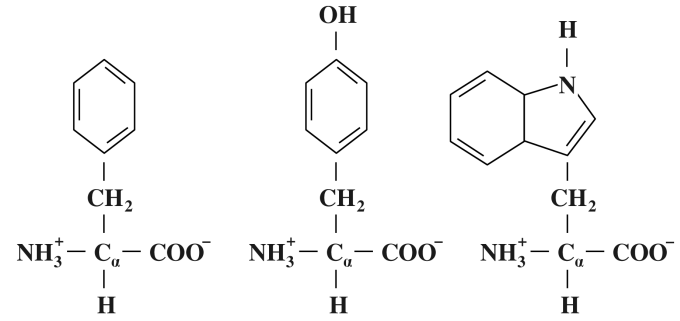


Рис. 9: Ароматические аминокислоты.

2.1.6 Аминокислоты, содержащие серу

Две аминокислоты — метионин и цистин — содержат атомы серы (Рис. 10). Эти аминокислоты гидрофобны и нейтральны. Интересным свойством цистина является то, что он может образовывать ковалентные дисульфидные связи с другими цистинами.

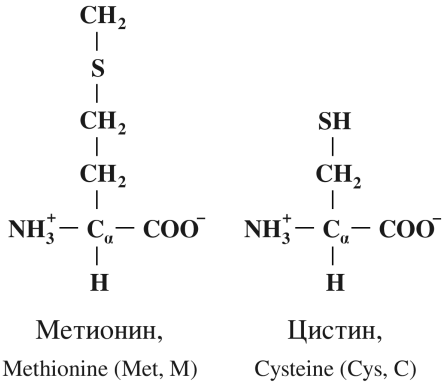
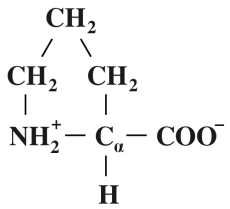


Рис. 10: Аминокислоты, содержащие серу.



Пролин,
Proline (Pro, P)

Рис. 11: Циклические аминокислоты.

2.1.7 Циклическая аминокислота

Пролин – единственная аминокислота, имеющая циклическую структуру – его боковой радикал ковалентно связан с амидной группой (Рис. 11). Из-за этого его гибкость крайне ограничена.

2.2 Первичная структура белков

Первичная структура определяет последовательность аминокислот, входящих в состав белка. Записывается первичная структура как последовательность однобуквенных или трёхбуквенных кодов аминокислот. Первичную структуру белка можно определить различными экспериментальными методами, включая метод секвенирования белков и методы определения первичной структуры кодирующей белок мРНК. Часто в последовательности аминокислот встречаются так называемые консервативные мотивы - устойчивые сочетания аминокислотных остатков, выполняющие определённую функцию и встречающиеся во многих белках. По степени сходства аминокислотных последовательностей белков из разных биологических видов можно оценивать эволюционные процессы и сходства между этими видами. Хотя первичная структура уникальна для конкретного белка, предсказать его функцию зная только первичную структуру практически невозможно.

Кроме первичной структуры выделяют ещё три уровня структур: вторичная, третичная и четвертичная. Вторичная структура представляет собой локальные конформации полипептидной цепи, такие как α-спирали и β-лучи. Некоторые комбинации вторичных

структур иногда также называют вторичной структурой или супервторичной структурой. Примером может служить β-шпилька, которая представляет собой два β-луча, соединённых коротким разворотом цепи белка. Полную трёхмерную структуру белка называют третичной структурой. Четвертичная структура описывает трёхмерную композицию индивидуальных доменов в больших мультидоменных белках.

2.3 Вторичная структура белков

2.3.1 α-спирали

α-спирали представляют собой плотные витки вокруг длинной оси молекулы, один виток составляет 3.6 аминокислотных остатка, шаг спирали составляет 0.54 нм (так что на один аминокислотный остаток приходится 0.15нм), спираль стабилизирована водородными связями между H и O пептидных групп, отстоящих друг от друга на 4 звена (Рис. 12). Шаблон водородных связей может быть записан как $(i, i+4)$, то есть каждая аминокислота связана водородной связью с аминокислотой на 4 впереди по цепи.

Кроме стандартной α-спирали, существует ещё два типа спиралей. Более плотная 3_{10}-спираль обладает характерным шаблоном водородных связей $(i, i+3)$. Такая спираль содержит 3 аминокислоты или примерно 10 тяжёлых атомов на виток. Более разряженная π-спираль обладает шаблоном водородных связей $(i, i+5)$. Ширина π-спирали такова, что через её внутреннюю полость может проходить молекула воды.

2.3.2 β-листы

В дополнение к α-структуре, β-структура является ещё одним очень распространённым типом локальной пространственной компоновки аминокислот. β-листы состоят из β-цепей, связанных с боков двумя или тремя водородными связям и образующих слегка закрученные складчатые листы. Более высокоуровневые структуры, образованные множеством β-листов, приводят к образованию белковых агрегатов и фибрилл. Такие структуры могут появляться при некоторых заболеваниях, в частности, амилоидозы (в том числе, болезнь

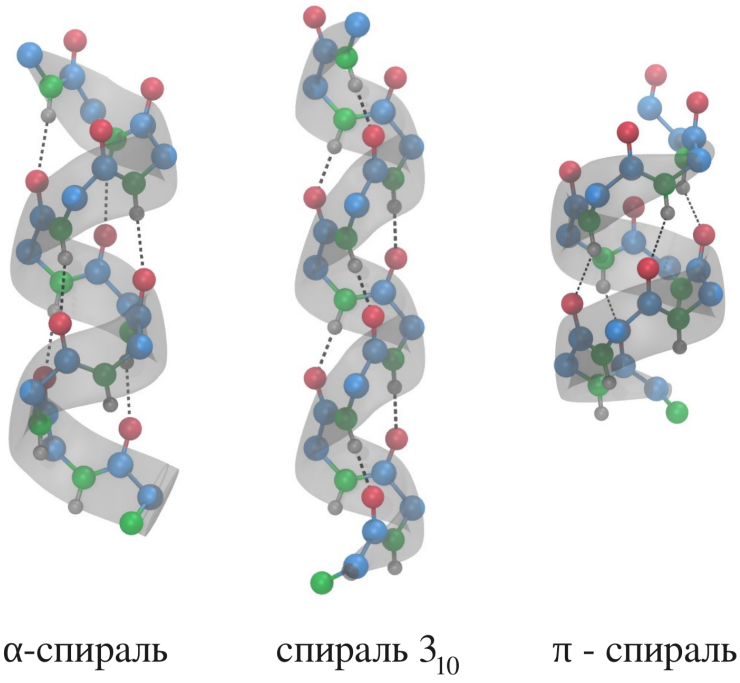


Рис. 12: Скелетные структуры α-спирали, спирали 3_{10} и π-спирали. Водородные связи между кислородом (красный) и амидной группой (зелёный и серый) показаны чёрными пунктирными линиями. α-спираль является средней по степени закрученности (3.6 аминокислот на виток). Для сравнения, в спирали 3_{10} на один виток приходится 3 аминокислоты, а в π-спирали их 4.1. Структуры синтезированы при помощи плагина "Molefacture" программы VMD [3].

Альцгеймера).

Возможны две различных компоновки β-цепей в β-листы: параллельная и антипараллельная (Рис. 13). В антипараллельном β-листе направления амидных связей полипептидных цепей противоположны. Элементарным антипараллельным β-листом можно считать β-шпильку, когда две β-цепи соединены коротким поворотом цепи (Рис. 14). Параллельный β-лист состоит из β-цепей, параллельных друг другу. В антипараллельном β-листе водородные связи более прочные, так как образующие их атомы расположены друг напротив друга.

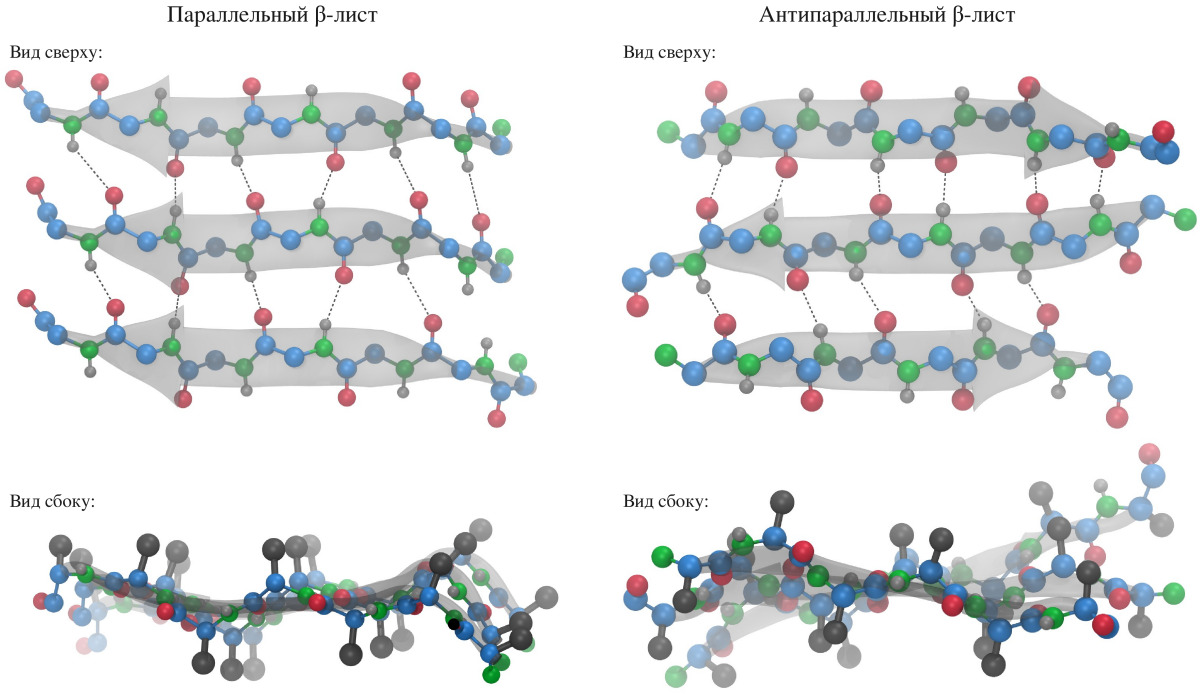


Рис. 13: Параллельный и антипараллельный β-листы. На виде сверху чёрными пунктирными линиями показаны водородные связи между атомами кислорода (красный) и амидными группами (зелёный и серый). На виде сбоку тёмно-серыми шарами показаны боковые радикалы аминокислот. Координаты атомов взяты из базы данных белковых структур (PDB код 2PEC для параллельного β-листа, 1IFB для антипараллельного).

2.3.3 Повороты и петли

Поворотами называются резкие изменения направления полипептидной цепи. Обычно повороты образуются между двумя антипараллельными β-цепями (например, в β-шпильке). По определению, две аминокислоты i и $i+1$ образуют поворот, если расстояние между аминокислотами $i-1$ и $i+2$ меньше, чем 7Å. Петли обычно длиннее поворотов (до пяти аминокислот) и соединяют другие типы вторичных структур, такие как α-спирали и параллельные β-листы. Повороты и петли обычно состоят из гидрофильных аминокислот и находятся на поверхности белка, то есть сильно погружены в растворитель.

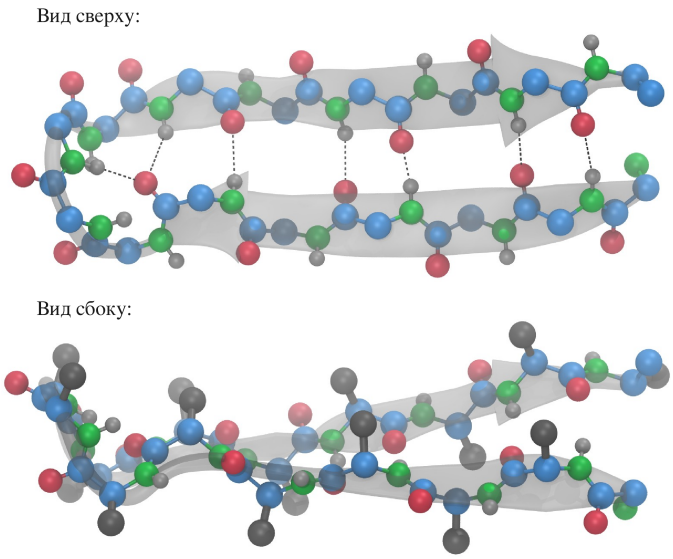


Рис. 14: Структура β-шпильки из белка GB1 (PDB код 1GB1), которая является примером элементарного антипараллельного β-листа. Водородные связи показаны при помощи пунктирных линий между кислородоми (красный) и амидными группами (зелёный и серый) полипептидного скелета. На виде сбоку показано расположение боковых радикалов.

2.4 Третичная структура белков

Третичная структура белка формируется из элементов вторичной структуры, таких как α-спирали и β-листы. Чёткой границы в определениии вторичной и третичной структуры белков не существует. Например, шпильки или структуры $\beta - \alpha - \beta$ можно отнести как к вторичной (супервторичной), так и к третичной структуре. Третичные структуры белков можно поделить на четыре класса:

1. α-спиральный класс белков, который включает в себя все белки, главным образом состоящие из α-спиралей, которые обычно образуют общее гидрофобное ядро. К этой категории можно отнести примерно 22% всех белков.

2. β-белки состоят в основном из β-цепей, сгруппированных в β-листы, стабилизированные множеством водородных связей. Эти белки обычно имеют несколько слоёв с общим гидрофобным ядром. Примерно 16% всех известных белков можно отнести к этому типу.

3. α/β-белки, которые состоят из перемежающихся α- и β-структур (примерно 15%).

4. $\alpha+\beta$-белки, в которых также присутствуют как α-, так и β-структуры, но в отличии от α/β-белков, в этой категории разные вторичные структуры пространственно удалены друг от друга.

Для дальнейшей структурной классификации белки могут разделяться на группы, имеющие похожие нативные структуры.

2.4.1 α-спиральный класс структур

Белки из α-спирального класса можно разделить на узлы, листы и массивы шпилек. α-спиральные узлы формируются, когда несколько α-спиралей собираются вокруг общего гидрофобного ядра. Примером такой третичной структуры может служить белок АСВР, который состоит из четырёх α-спиралей (Рис. 15). α-спиральный лист присутствует в нативном состоянии миоглобина (Рис. 15), в котором восемь α-спиралей образуют многоуровневую структуру. Третичная структура миоглобина стабилизирована главным образом сильными гидрофобными взаимодействиями. Существуют и другие типы α-структур.

2.4.2 β-белки

β-белки можно также разделить на несколько типов. Первый и самый распространённый - β-сэндвич. Этот тип структуры присутствует, например, в нативной структуре иммуноглобулярных доменов титина (Рис. 16). Стабилизируется β-сэндвич гидрофобными взаимодействиями между β-листами. β-бочка – ещё один пример β-белков, в котором β-лист завёрнут так, что крайние β-лучи также образуют водородные связи. Пример такого белка приведён на рисунке 16. Ещё одним интересным примером β-белков может служить β-спираль (Рис. 16). Вероятно, такие структуры могут появляться в амилоидных фибрилах.

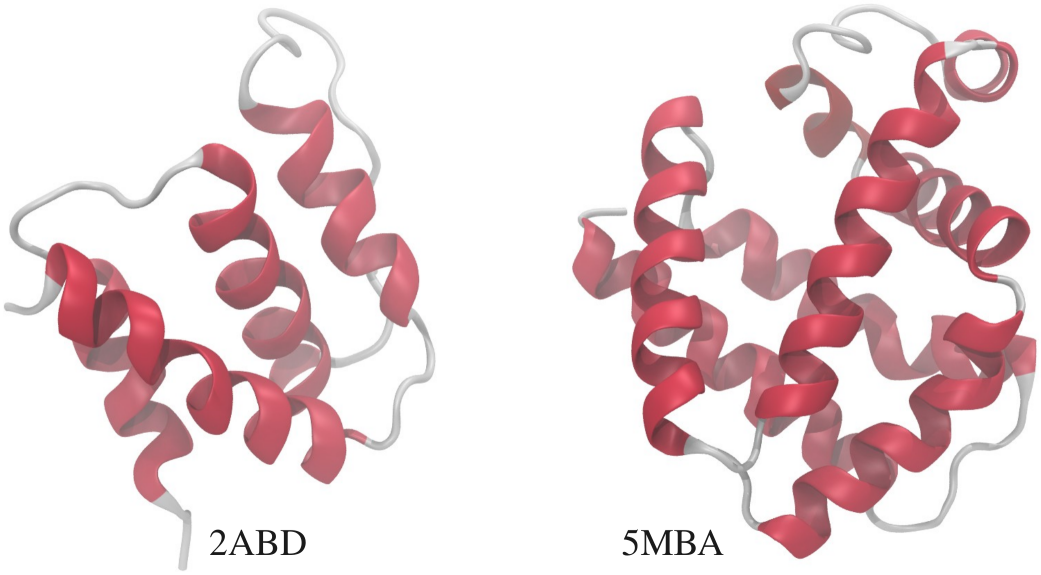


Рис. 15: Нативные структуры белка ACBP (PDB код 2ABD) как образец спирального узла и миоглобина (PDB код 5MBA) как образец α-спирального листа.

2.4.3 α/β и $\alpha+\beta$-белки

Большое число структур можно отнести к классам α/β и $\alpha+\beta$. Например, нативная структура изомеразы триосефосфата представляет из себя одну очень часто встречающуюся α/β-структуру - α/β-бочку, в которой α-спирали и β-лучи перемежаются и закручиваются, образуя форму, напоминающую бочку (Рис. 17). Простейшую структуру из класса $\alpha+\beta$-белков образует домен B1 белка G (Рис. 17). В этой структуре α-спираль прижата к состоящему из четырёх цепей β-листу. Ещё одним примером $\alpha+\beta$-белка может служить нативная структура лизосомы, в которой небольшой β-лист расположен на стороне большого α-спирального домена (Рис. 17).

2.5 Нативное состояние белков

Нативное состояние белка стабилизируется электростатическими взаимодействиями и взаимодействиями Ван-дер-Ваальса между атомами белка, а также между атомами белка и растворителя. Сложный баланс сил между атомами определяет нативную структуру

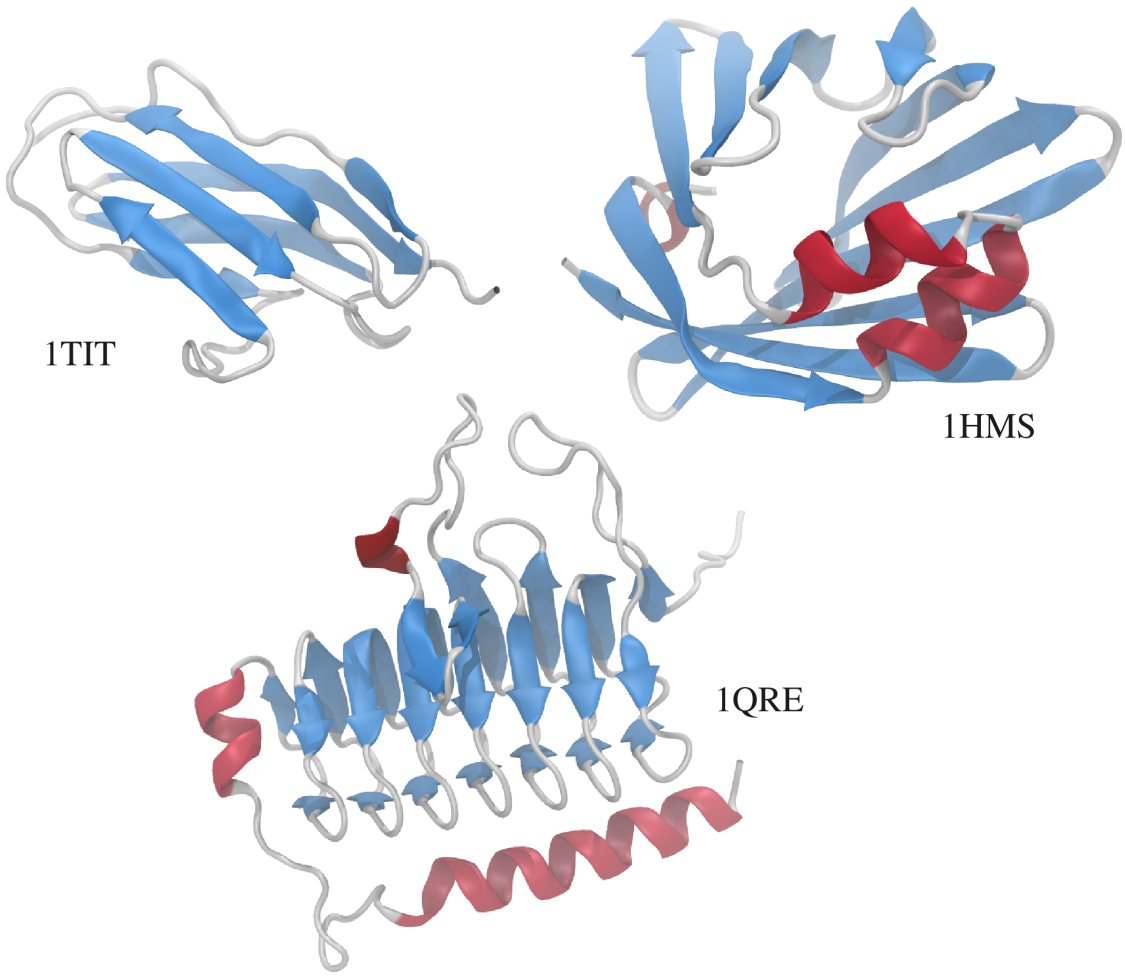


Рис. 16: Нативные структуры: домена титина Ig27 (PDB код 1TIT) представляющего из себя β-сэндвич, белка связывания жирных кислот (PDB код 1HMS), который формирует β-бочку и углеродной ангидразы (PDB код 1QRE) как образец β-спирали.

белка:

1. Отталкивания, исключающие взаимное проникновение объёмов Ван-дер-Ваальса двух атомов.

2. Солевые мостики или электростатические взаимодействия между заряженными аминокислотами.

3. Водородные связи, которые являются электростатическими и возникают в результате

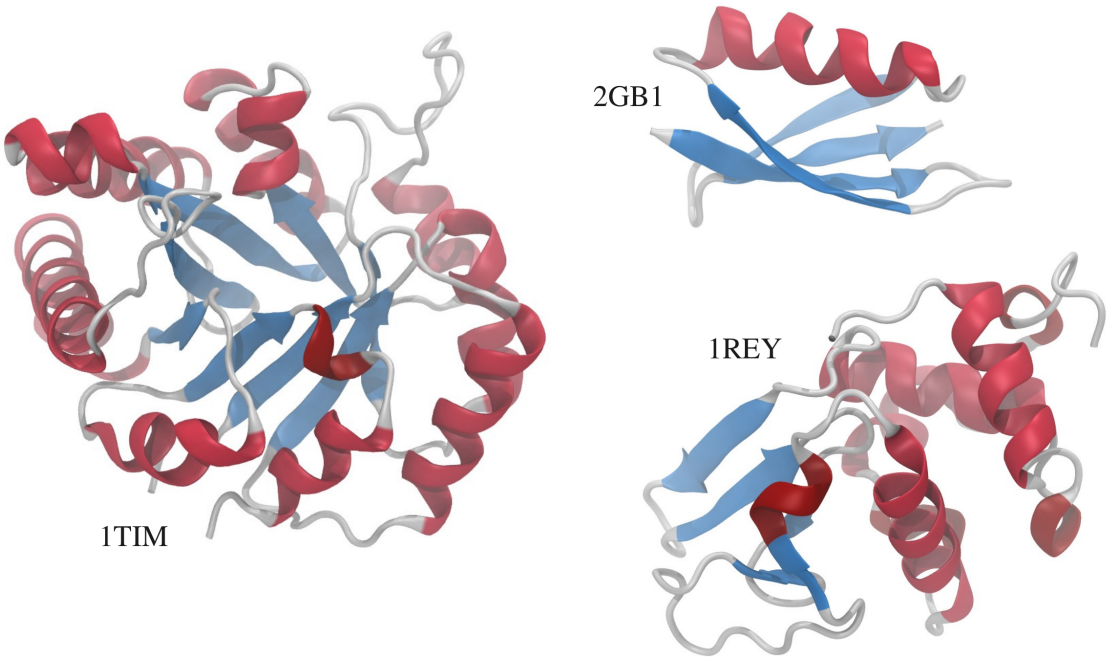


Рис. 17: Изомераза триосефосфата (PDB код 1TIM) образует α/β-бочку, белок GB1 (PDB код 2GB1), принадлежащий к $\alpha + \beta$-классу и лизосома (PDB код 1REY) как представитель $\alpha + \beta$-класса.

разделения водорода между донором и акцептором.

4. Гидрофобные взаимодействия, которые являются следствием того, что гидрофобные аминокислоты обладают меньшей свободной энергией, будучи внутри белка, а гидрофильные — будучи снаружи. Основная причина гидрофобных взаимодействий – взаимодействие между белком и растворителем.

Белки синтезируются рибосомой, которая считывает генетическую информацию с ДНК. После этого белки попадают в клеточную среду и образуют нативную структуру. Самоорганизацией или фолдингом белка называется процесс сворачивания белка в котором ковалентно связанная последовательность аминокислот образует нативную структуру белка. Экспериментально показано, что белки могут денатурировать и образовывать нативное состояние без внешнего вмешательства. В физиологических условиях среды, нативное состояние белка определяется исключительно последовательностью аминокислот. Согласно

текущей теории фолдинга, в нативном состоянии белка достигается минимум свободной энергии для полипептидной цепи белка.

Для того, чтобы понять проблемы обнаружения механизма фолдинга белков (или проблему фолдинга), можно рассмотреть парадокс Левинталя (Cyrus Levinthal), сформулированный в 1967 году [4]. Рассмотрим белок, состоящий из $N = 100$ аминокислот. Так как аминокислоты могут вращаться друг относительно друга, каждая из аминокислот может принять одну из как минимум 10 конформаций. В таком случае, общее число конформаций, доступных белку, составляет $C = 10^N = 10^{100}$. Будем считать, что скорость смены конформации порядка $s = 10^{14}$ конформаций в секунду (это примерно соответствует частоте столкновений белка с молекулами водного раствора). Если фолдинг белка – случайный процесс, в котором все конформации «апробируются», то время фолдинга белка составит $\tau_F \sim C/s \sim 10^{80}$ лет, что превышает возраст вселенной. Поэтому, должен быть какой-то направленный путь от растянутой полипептидной цепи к нативному состоянию.

2.5.1 "Старый" взгляд на проблему фолдинга

Парадокс Левинталя может быть разрешён, если считать, что белок образует нативное состояние, проходя через несколько хорошо определённых состояний. Эта теория была выдвинута в процессе изучения фолдинга белка BPTI. Этот белок состоит из 56 аминокислот, среди которых – 6 цистинов (Рис. 18), которые могут образовывать дисульфидные связи. В 70-х годах прошлого века Томасом Креигтоном (Thomas Creighton) было показано, что у BPTI существует три промежуточных состояния, через которые он проходит на пути к нативной структуре [5]. Эти состояния характеризуются последовательно формируемыми тремя дисульфидными связями. Креигтоном предположил, что процесс фолдинга любого белка может быть представлен в виде последовательного перехода между промежуточными состояниями, а кинетическая реакция фолдинга может быть представлена в виде следующего уравнения:

$$U \rightleftharpoons I_1 \rightleftharpoons I_2 \rightleftharpoons \ldots \rightleftharpoons F, \tag{1}$$

где U – денатурированое состояние, F – нативное состояние, а I_n – промежуточные состояния, которые характеризуются конфигурацией дисульфидных связей.

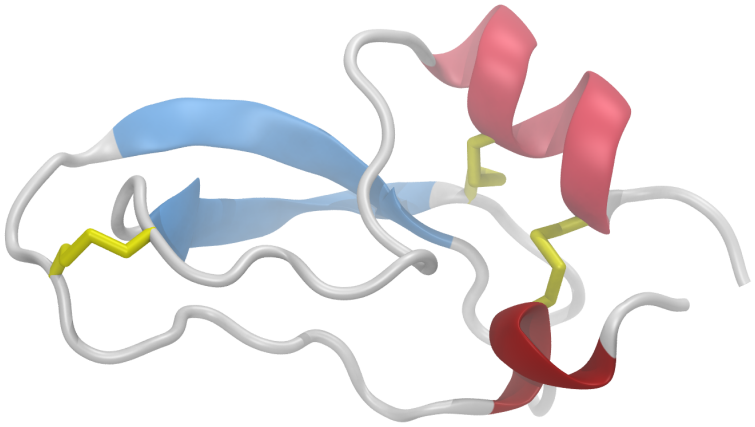


Рис. 18: Нативное состояние белка BPTI (PDB код 1G6X), включающее в себя три дисульфидные связи между цистинами (показанны оранжевым цветом).

2.5.2 "Новый" взгляд на проблему фолдинга

Экспериментально было показано, что процесс фолдинга даже небольшого белка ($N < 100$) сильно направленн и без каких-либо хорошо определённых промежуточных состояний. Таким образом, теория выдвинутая Креигтоном в общем случае оказалась не верной. Результаты одного из экспериментов по исследованию процесса фолдинга белка показаны на рисунке 19.

Как видно из рисунка 19, при охлажеднии раствора белка и при прохождении точки плавления, происходит резкий переход в нативное состояние. То есть все молекулы белка в растворе обретают нативную структуру, а переход из денатурированного состояния в нативное происходит в один шаг, без промежуточных состояний. Таким образом, кинетическое уравнение процесса фолдинга можно записать как:

$$U \rightleftharpoons F. \tag{2}$$

Энергетический ландшафт свободной энергии удобно представить в виде воронки фол-

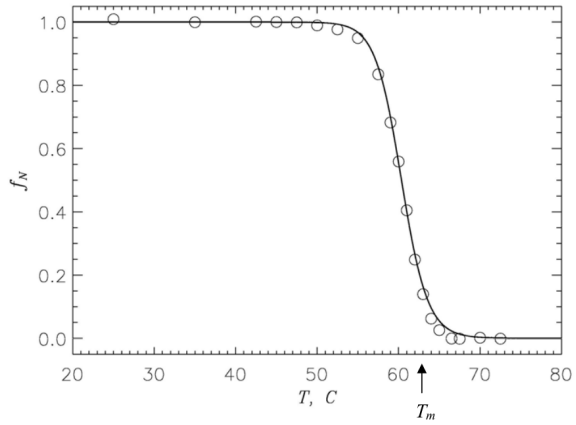


Рис. 19: Температурный фолдинг белка, показанный как доля белка в растворе, находящаяся нативном состоянии (f_N). Резкий переход в районе «точки плавления» (T_m) показывает, что фолдинг происходит в один шаг.

динга (Рис. 20). Данное представление носит исключительно качественный характер — изображение воронки фолдинга для реального белка невозможно в силу большого числа степеней свободы ($3 \times N$, где N-число атомов в белке). Белок начинает свой путь к нативному состоянию из вытянутой структуры. Эта конформация обладает большой энтропией и энергией. В «горлышке» воронки — нативное состояние, которое соответствует минимуму свободной энергии. После синтеза, белок начинает своё движение вниз по воронке. При этом уменьшается и энтропия и энергия системы. Если по ходу сворачивания цепь должна близко подойти к своей нативной структуре перед тем, как начнут возникать стабилизирующие её контакты, то повышение свободной энергии цепи будет большим за счёт резкого уменьшения энтропии. Такой процесс будет очень медленным. Если же путь сворачивания таков, что падение энтропии компенсируется падением энергии, то процесс фолдинга будет быстрым.

Теоритическое изучение проблемы фолдинга белков практически с самого начала ориентировалось на использование вычислительного эксперимента. В ранних работах пытались максимально точно учитывать взаимодействия между элементами белка. Однако, до недавнего времени, произвести подобные рассчёты не представлялось возможным в силу их вычислительной сложности. На сегодняшний день, структуры нескольких белков были

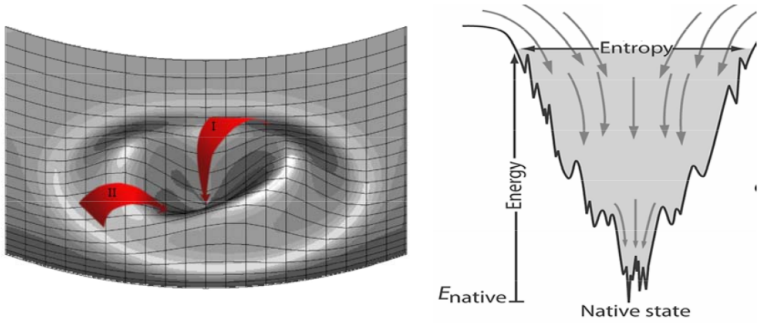


Рис. 20: Схематическое изображение ландшафта свободной энергии (слева) и его версия с указанием различных энергетических составляющих (справа). Свободная энергия минимальна на дне воронки, что соответствует нативному состоянию. Барьеры свободной энергии показаны как неровности на поверхности. Денатурированые состояния находятся на верхнем, широком, конце воронки. Сужение воронки при приближении к нативному состоянию показывает уменьшение энтропии, что компенсируется уменьшением энергии (то есть увеличением модуля притягательных взаимодействий). Красные стрелки показывают различные пути к нативному состоянию.

получены из последовательности аминокислот при помощи вычислительного эксперимента, то есть путём их моделирования на компьютере [6].

3 Нуклеотиды, ДНК и РНК

Нуклеиновые кислоты – высокомолекулярные органические соединения, биологические молекулы, состоящие из связанных ковалентно нуклеотидов. В зависимости от входящих в их состав нуклеотидов, в клетках живых организмов выделяют два основных вида нуклеиновых кислот – дезоксирибонуклеиновую и рибонуклеиновую кислоты (ДНК и РНК соответственно). Основной задачей этой группы биологических молекул является хранение, передача и использование наследственной информации организмов. Открытие нуклеотидов датируется 1869 годом, когда швейцарский учёный Фридерик Мишер (Friedrich Miescher) впервые изолировал нуклеиновые кислоты из ядер клеток [7]. Само название "нуклеиновые кислоты" произошло от английского nuclei (ядро). Изначально считалось, что полученный Мишером материал служит хранилищем фосфора и учёные не уделили ему должного интереса. Только после 1944 года, когда была показана важность нуклеиновых кислот для хранения и передачи информации [8], начался новый виток изучения данных молекул. Важнейшее открытие в данной области не заставило себя ждать и в 1953 году Джеймс Уотсон (James Watson) и Френсис Крик (Francis Crick) впервые представили модель структуры ДНК в виде двойной спирали [9]. Параллельно с Уотсоном и Криком над разрешением структуры ДНК работали Розалинд Франклин (Rosalind Franklin) и Морис Уилкинсон (Maurice Wilkins), которые занимались рентгеновской кристаллографией. Розалинд, которая была очень аккуратна и боялась опубликовать неверные результаты, к тому времени уже получила рентгеновские снимки молекулы ДНК. Зная о её работе, Уотсон предложил сотрудничество, но Франклин отказалась. Несмотря на это, Уилкинсон показал её снимки Уотсону, что помогло последнему построить конечный вариант модели. Обе научные группы опубликовали результаты практически одновременно – в первой половине 1953 года, лишь "мелким текстом" ссылаясь на работы друг друга [9, 10]. В конечном итоге, Уотсон, Крик и Уилкинсон получили Нобелевскую премию. Несомненно, в этом списке была бы и Франклин, чьи заслуги в разрешении структуры сложно переоценить, но она к тому времени скончалась от рака, а Нобелевские премии посмертно не вручаются. С тех пор в научно-популярной и образовательной литературе, когда говорят об открытии

структуры ДНК вспоминают Уотсона и Крика, а о работе Франклин зачастую забывают.

3.1 Нуклеотиды

Нуклеотиды состоят из фосфатной группы (остатка фосфорной кислоты), нуклеозида (сахара) и азотистого основания (Рис.21 и 22). Между собой нуклеотиды различаются химическим составом оснований, а также видом сахара. Выделяют пять основных азотистых оснований – аденин, гуанин, тимин, цитозин и урацил (Рис. 21). Первые четыре из них встречаются в молекуле дезоксирибонуклеиновой кислоты (ДНК), в то время как в составе рибонуклеиновой кислоты (РНК) обычно нет тимина, но есть урацил. Кроме того, нуклеотиды, входящие в состав ДНК и РНК, различаются химическим составом сахара. Интересно, что молекула аденозинтрифосфата (АТФ), являющаяся одним из основных носителей энергии в организме, также является нуклеотидом (с аденином в качестве основания, Рис. 22). По аналогии с белками, последовательность входящих в состав ДНК и РНК нуклеотидов часто называют их первичной структурой.

3.2 ДНК

Несмотря на столь малые различия в химической формуле нуклеотидов, входящих в состав ДНК и РНК, пространственная структура этих макромолекул существенно отличается. У подавляющего большинства организмов вторичная структура ДНК представляет собой не одну, а две полинуклеиновых цепи. Эти две длинные цепи закручены одна вокруг другой в виде двойной спирали, стабилизированной водородными связями, образующимися между обращёнными друг к другу азотистыми основаниями противоположных цепей (Рис. 23). Каждое из оснований одной из цепей имеет комплиментарную пару – конкретное основание в противоположной цепи: аденин образует связи только с тимином, а цитозин – с гуанином. Кроме того, спираль также стабилизируется гидрофобными взаимодействиями и силами стэкинга, возникающими между плоскостями азотистых оснований последовательных нуклеотидов. Комплиментарность оснований означает, что информация, содержащаяся в одной из цепей, содержится и в другой. Дублирование информации дела-

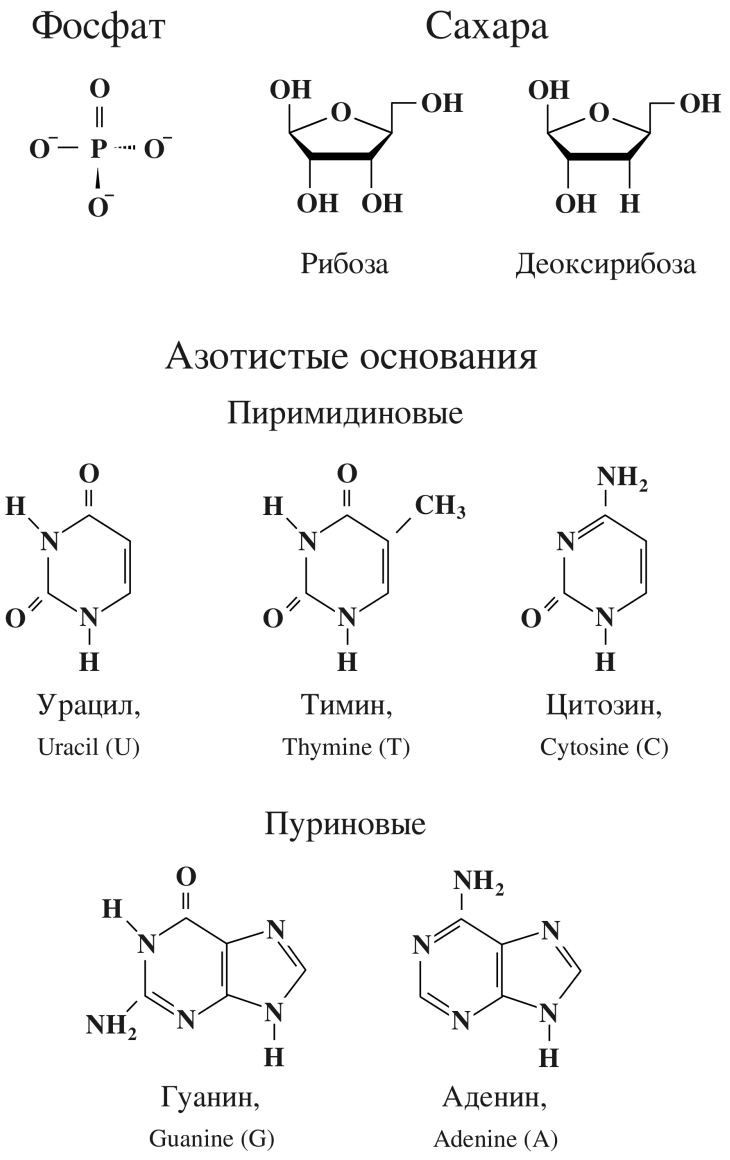


Рис. 21: Составные части нуклеотидов: фосфатные группы, сахар и азотистые основания. Для последних также указаны английские названия и однобуквенные обозначения, которые используются в записи первичной структуры ДНК.

ет молекулу ДНК идеальной для хранения генетической информации и её передачи при делении клеток.

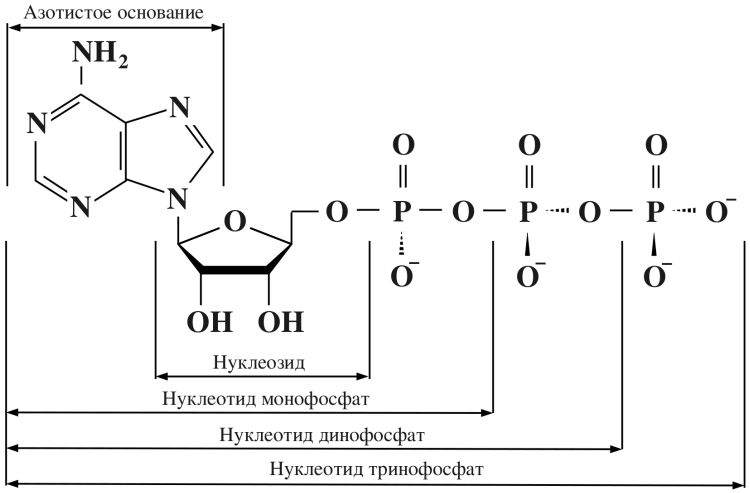


Рис. 22: Аденозин трифосфат как пример нуклеотида. На рисунке выделены составные части молекулы: нуклеотид (в данном случае Аденин), сахар (рибоза) и три фосфатные группы. В молекуле ДНК нуклеотиды бывают четырёх видов (аденин, гуанин, тимин, цитозин), в качестве сахара выступает дезоксирибоза, а фосфатная группа – одна.

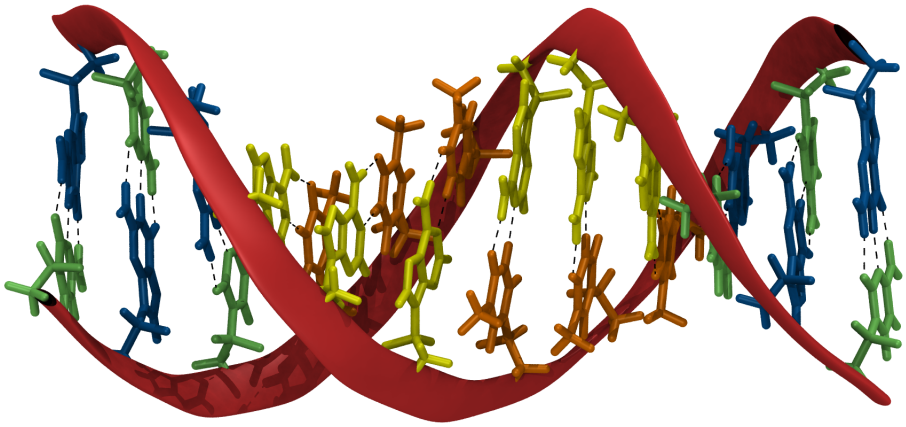


Рис. 23: Схема двойной спирали ДНК. Две цепи нуклеотидов, расположеных антипараллельно, взаимодействуют между собой посредством водородных связей (чёрные пунктирные линии). В центре спирали оказываются азотистые основания: гуанин (синий), цитозин (зелёный), аденин (жёлтый) и тимин (оранжевый).

3.3 РНК

В отличие от ДНК, молекулы РНК преимущественно существуют в виде одной полинуклеотидной цепи. Это объясняется тем, что наличие дополнительной группы OH дела-

ет несколько другую пространственную конформацию энергетически выгодной. При этом РНК могут выполнять целый ряд функций в организме – от хранения информации до синтеза белков и регуляции генов. Часто молекулы РНК обладают ярко выраженной третичной структурой. В некотором смысле РНК является универсальной молекулой, так как она может выступать как в роли ДНК, так и в роли белка. Последнее можно проиллюстрировать примером синтеза белка, в котором участвует три разных типа РНК. Информация о последовательности аминокислот в синтезируемом белке содержится в матричной рибонуклеиновой кислоте (мРНК) – три последовательных нуклеотида, называемых кодоном, кодируют одну аминокислоту. Транспортные РНК (тРНК), состоящие из менее чем 100 нуклеотидов, переносят специфические аминокислоты к месту синтеза – рибосоме. Последняя состоит, в основном, из рибосомальных РНК (рРНК). Рибосома присоединяется к мРНК, считывает информацию, одновременно синтезируя белок из доставленных молекулами тРНК аминокислот. В данном процессе молекулы РНК выполняют роль промежуточного хранилища информации, а также наделены функцией транспорта аминокислот и контроля над биологической реакцией синтеза белка.

Часть II

Молекулярное моделирование

4 Предпосылки

Рентгеновская кристаллография может предоставить неподвижную картину взаимного расположения аминокислот в нативном состоянии белка. Однако в физиологической среде большинство белков могут менять свою структуру благодаря локальным движениям атомов белка под действием обычной внешней среды (нормальная кислотность, температура, давление, ионная сила). Более того, во многих случаях локальные структурные изменения (например, переход из одного подсостояния в другое) могут играть столь же важную роль в биологической функции белка, что и глобальные конформационные изменения (такие как диссоциация белок-белковых комплексов или денатурация белков). В некоторых случаях удаётся получить несколько промежуточных структур, кристаллизуя белок при различных условиях среды или используя методы Ядерного Магнитного Резонанса (ЯМР). Но даже в таких случаях получить информацию о механизме того или иного конформационного изменения, его кинетических и термодинамических свойствах практически невозможно. Вычислительные методы, такие как Молекулярная Динамика (МД) и Динамика Ланжевена (ДЛ), способны заполнить этот пробел, предоставив динамическую картину микромолекулярных движений, дополнив экспериментальные результаты, предоставив максимально детальные знания о динамике биологических молекул, движении их структурных и функциональных фрагментов, поведении элементов вторичной структуры.

5 Молекулярная механика

Основной идеей молекулярной механики является то, что молекулярная система может быть рассмотрена как микроскопическая механическая система. Согласно этой идее, все атомы в системе связаны механическими силами, которые контролируют длину ковалентных связей, углы, образуемые ковалентными связями, вращения вокруг связей и так далее (Рис. 24). Атомы взаимодействуют между собой согласно классическим невалентным потенциалам, которые определяют невалентные атомарные силы. Математически, это описание требует составления потенциальной функции, которая включает в себя исключительно классические величины. Эта функция потенциальной энергии, называемая силовым полем, затем может быть использована для вычисления сил взаимодействия между атомами, подставив которые в уравнения движения (напр. уравнения Ньютона) можно описать движение атомов системы с течением времени. Существует три фундаментальных принци-

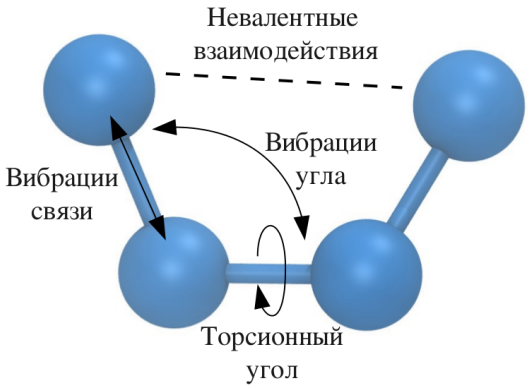


Рис. 24: Идея молекулярной механики, проиллюстрированная при помощи фрагмента молекулы в 4 атома. Четыре типа взаимодействий показывают основные члены функции потенциальной энергии системы.

па молекулярной механики: термодинамическая гипотеза, аддитивность и переносимость.

1. Термодинамическая гипотеза заявляет, что нативное состояние белка соответствует минимуму функции потенциальной энергии. Уникальное нативное состояние белка закодировано в последовательности аминокислот, потому что сложная топология ковалентных и нековалентных взаимодействий ведёт белок в сторону его нативного

состояния, которое соответствует структуре с минимальной энергией. Это же предположение лежит в основе “нового” взгляда на проблему фолдинга. Хотя это предположение нельзя точно доказать, экспериментальные и теоретические результаты его поддерживают.

2. Аддитивность предполагает, что полная потенциальная энергия может быть разложена в сумму членов, описывающих невалентные взаимодействия (электростатические и взаимодействия Ван-дер-Ваальса) и валентные взаимодействия между атомами белка. То есть связь между всеми этими слагаемыми отсутствуют и все они могут быть посчитаны независимо. Это предположение в общем случае не верно. Например электростатические взаимодействия между двумя атомами могут повлиять на распределение зарядов на других атомах (через эффект поляризации). Поэтому в уравнение для потенциальной энергии должны также входить и взаимодействия между многими атомами. Существуют силовые поля, которые учитывают эффект поляризации. Последние возникли в процессе решения задачи фолдинга: оказалось, что не все белки сворачиваются в нативную структуру при моделировании с использованием обычных силовых полей. С другой стороны, принцип аддитивности позволяет существенно ускорить вычислительный процесс и в большинстве случаев эффектом поляризации пренебрегают.

3. Гипотеза о переносимости предполагает, что свойства атомов в большой молекуле может быть получено через изучение аналогичных атомов в аналогичном окружении, но в маленьких молекулах. Таким образом, все возможные взаимодействия между атомами можно обобщить в конечную выборку, для которой необходимо задать параметры потенциалов взаимодействия. Эти параметры затем могут быть использованы при моделировании большой молекулы. Этот подход применяется в разработке всех современных функций потенциальной энергии.

6 Силовое поле

Связь между координатами атомов биомолекулы и её энергией является основной частью вычислительных моделей атомарных систем. Функция потенциальной энергии (силовое поле, англ. force-field) [11, 12] используется для того, чтобы вычислить энергию системы, а также определить силы взаимодействия между входящими в неё атомами. Силовое поле задаётся функцией от координат атомов системы (функциональная часть силового поля), зависящей от некоторого набора параметров (параметрическая часть).

6.1 Функциональная часть силового поля

Опираясь на принцип аддитивности, функциональная часть силового поля V обычно задаётся суммой ковалентных (V_b) и невалентных взаимодействий (V_{nb}) в системе [13–17]:

$$V = V_b + V_{nb}, \tag{3}$$

где ковалентные взаимодействия описываются гармоническими вибрациями ковалентных связей (англ. bonds), углов между тремя атомами (англ. angles) и двумя разными типами торсионных взаимодействий – для правильных (англ. dihedrals) и неправильных торсионных углов (англ. improper dihedrals):

$$V_b = \sum_{bonds} \frac{1}{2} K_s (b - b_0)^2 + \sum_{angles} K_\theta (\theta - \theta_0)^2 + \sum_{dihedrals} K_\phi (1 - cos(n\phi - \phi_0)) + \sum_{impropers} K_\psi (\psi - \psi_0)^2, \tag{4}$$

где b и θ – расстояние между двумя атомами и угол между двумя смежными ковалентными связями, ϕ и ψ – торсионные углы поворота двух связей вокруг третьей, смежной к ним. b_0, θ_0, ϕ_0, и ψ_0 – равновесные значения этих величин. Эти значения рассчитываются из текущего расположения атомов в системе (их пространственных координат). K_s, K_θ, K_ϕ – коэффициенты упругости вибрации длины ковалентной связи, угла между двумя смежными связями, правильных и неправильных торсионных вращений. Важно заметить, что данные параметры зависят от типов атомов и задаются изначально для каждого кон-

кретного силового поля (параметрическая часть силового поля) [13, 18]. Это, теоретически, позволяет производить молекулярное моделирование для любой системы, вне зависимости от того, разрешена ли её кристаллическая структура. В некоторых силовых полях формула (4) дополняется и другими членами, например, корректировками Урея-Брэдли[13] и/или корректировками CMAP для торсионных углов [19].

Нековалентная часть функции потенциальной энергии чаще всего состоит из потенциала электростатических взаимодействий и потенциала Ван-дер-Ваальса:

$$V_{nb} = \sum_{i,j} \left(\frac{q_i q_j}{4\pi\varepsilon_0 \varepsilon r_{ij}} + \varepsilon_{ij} \left[\left(\frac{\sigma_{ij}}{r_{ij}} \right)^{12} - 2 \left(\frac{\sigma_{ij}}{r_{ij}} \right)^{6} \right] \right) \tag{5}$$

где r_{ij} – расстояние между двумя взаимодействующими атомами, q_i и q_j – заряды атомов; ε и ε_0 – диэлектрическая проницаемость среды и диэлектрическая постоянная; $\varepsilon_{ij} = \sqrt{\varepsilon_i \varepsilon_j}$ и $\sigma_{ij} = (\sigma_i + \sigma_j)/2$ – параметры Ван-дер-Ваальса для атомов i и j. Изначально, функция потенциальной энергии также явно включала член, описывающий водородные связи [13], но позднее от него отказались, неявно включив данные взаимодействия в электростатический потенциал при помощи частичных зарядов на донорах и акцепторах водородных связей [14].

Так как биологические молекулы находятся в водной среде, необходимо учитывать влияние водного окружения – гидрофобный эффект, экранирование электростатических взаимодействий водной средой и так далее. С точки зрения методов вычисления, самый простой способ – явное добавление молекул воды в систему. Такой способ называется моделированием в явно заданном растворителе, и на сегодняшний день это один из самых популярных и точных методов. Добавление воды существенно увеличивает количество атомов, для которых необходимо производить расчёт – добавленные в систему молекулы воды участвуют в электростатических взаимодействиях, взаимодействиях Ван-дер-Ваальса, их надо перемещать согласно заданным уравнениям движения. Кроме того, чтобы ограничить количество атомов в системе, необходимо вводить граничные условия, которые бы не позволили воде удаляться от молекулы.

Существуют также модели, неявно описывающие взаимодействие с окружающей средой (с водой) [20]. Для этого потенциальная функция расширяется дополнительным членом, имитирующим гидрофобный эффект и заземление электростатики, а механические столкновения с молекулами воды моделируются при помощи случайной силы – уравнения Ньютона заменяются уравнениями Ланжевена. Самым популярным подходом для моделирования гидрофобного эффекта является энергетическая функция, пропорциональная площади атомов, доступной растворителю (модель SASA) [21]. Электростатические эффекты чаще всего описываются при помощи обобщённой теории Борна [22]. При моделировании в неявном растворителе общее количество степеней свободы уменьшается, в зависимости от размера системы, примерно в 10 раз, если сравнивать с моделированием в явном растворителе. Но, так как используется схожая форма потенциальной функции, шаг по времени по прежнему не может превышать 1–2 фс.

6.2 Параметры силового поля

Согласно гипотезе о переносимости, свойства атомов зависят от его локального окружения, такого как характер ковалентных связей или состояние гибридизации. Например, свойства атома углерода зависит от того, является ли этот атом частью ароматической или алифатической цепи. Аналогично, атомы водорода могут быть как полярными так и неполярными, в зависимости от локального окружения. Для того, чтобы задать параметры функции силового поля, используются типы атомов, определяющие локальное окружение рассматриваемого атома и, как следствие, его свойства и параметры. Одно из стандартных силовых полей CHARMM27 использует более 150 различных типов атомов: для описания белков в этом силовом поле используются 83 различных типов углерода, 17 разных типов водородов, 43 разных типов азотов и так далее. Для каждого типа атома и их комбинаций силовым полем задаётся необходимый для расчёта потенциальной энергии набор параметров (параметры Ван-дер-Ваальса ε_i и σ_i, равновесные значения b_0, θ_0, ϕ_0, и ψ_0, коэффициенты упругости K_s, K_θ и K_ϕ и т.д.).

Файл с параметрами силовых полей и описанием топологий в формате программ CHARMM

и NAMD можно скачать с сайта Алекса МакКерела (Alex MacKerell[1]). Фрагмент файла топологий, описывающий типы атомов в силовом поле CHARMM27 (“top_all27_prot_na.rtf”) выглядит следующим образом:

```
MASS     1 H      1.00800 H ! polar H
MASS     2 HC     1.00800 H ! N-ter H
MASS     3 HA     1.00800 H ! nonpolar H
MASS     4 HT     1.00800 H ! TIPS3P WATER HYDROGEN
MASS     5 HP     1.00800 H ! aromatic H
MASS     6 HB     1.00800 H ! backbone H
...
MASS    20 C     12.01100 C ! carbonyl C, peptide backbone
MASS    21 CA    12.01100 C ! aromatic C
MASS    22 CT1   12.01100 C ! aliphatic sp3 C for CH
MASS    23 CT2   12.01100 C ! aliphatic sp3 C for CH2
MASS    24 CT3   12.01100 C ! aliphatic sp3 C for CH3
...
MASS    54 NH1   14.00700 N ! peptide nitrogen
MASS    55 NH2   14.00700 N ! amide nitrogen
MASS    56 NH3   14.00700 N ! ammonium nitrogen
...
MASS    70 O     15.99900 O ! carbonyl oxygen
MASS    71 OB    15.99900 O ! carbonyl oxygen in acetic acid
MASS    72 OC    15.99900 O ! carboxylate oxygen
MASS    73 OH1   15.99900 O ! hydroxyl oxygen
MASS    74 OS    15.99940 O ! ester oxygen
MASS    75 OT    15.99940 O ! TIPS3P WATER OXYGEN
```

Атом водорода H может быть как полярными (типа атома H), каким он является в амидной связи, так и неполярным (типа атома HA), что используется в боковых радикалах

[1]http://mackerell.umaryland.edu/

аминокислот валин и аланин (Рис. 5). Для каждого типа в данном фрагменте приведена масса и краткий комментарий. Необходимо различать типы атомов и их имена (которые показаны в файле координат). Имена атомов должны быть уникальны для конкретной аминокислоты, в то время как атом одного и того же типа может быть найден во множестве разных аминокислот. Стоит также обратить внимание на то, что для каждого типа атома задаётся набор параметров Ван-дер-Ваальса, в то время как его частичный заряд зависит от локального окружения атома. Например, углероды C_α и C_β в валине представлены в виде одного и того же алифатического типа CT1, но, несмотря на это, обладают различными зарядами. Соответствие атомов в аминокислотах типам и их частичный заряд задаётся в том же файле. Описание всех атомов аминокислоты валин выглядит следующим образом:

```
1  RESI VAL          0.00
2  GROUP
3  ATOM N    NH1    -0.47  !      |    HG11 HG12
4  ATOM HN   H       0.31  !  HN-N       | /
5  ATOM CA   CT1     0.07  !      |     CG1--HG13
6  ATOM HA   HB      0.09  !      |    /
7  GROUP                   !  HA-CA--CB-HB
8  ATOM CB   CT1    -0.09  !      |    \
9  ATOM HB   HA      0.09  !      |     CG2--HG21
10 GROUP                   !   O=C    / \
11 ATOM CG1  CT3    -0.27  !      | HG21 HG22
12 ATOM HG11 HA      0.09
13 ATOM HG12 HA      0.09
14 ATOM HG13 HA      0.09
15 GROUP
16 ATOM CG2  CT3    -0.27
17 ATOM HG21 HA      0.09
18 ATOM HG22 HA      0.09
```

```
19 ATOM HG23 HA       0.09
20 GROUP
21 ATOM C    C        0.51
22 ATOM O    O       -0.51
```

Все атомы аминокислоты разделены на группы с целым частичным зарядом (чаще всего нулевым). Последний для каждого атома задан в четвёртой колонке. Видно, что для атома C_α (CA) частичный заряд равен 0.07, в то время как для C_β (CB) он равен -0.09 элементарных зарядов. Далее в файле топологий перечислены все ковалентные связи для аминокислоты валин, а также наборы атомов, для которых следует рассчитывать потенциал углов, потенциал торсионных углов и так далее. Таким образом, зная последовательность аминокислот в белке, можно задать связанность атомов в молекуле (какой атом взаимодействует с каким и посредством какого потенциала), узнать типы всех атомов в системе и их частичные заряды. Далее рассмотрим более подробно все слагаемые потенциальной функции и то, как задаются соответствующие параметры в силовом поле CHARMM27.

6.2.1 Потенциал ковалентной связи

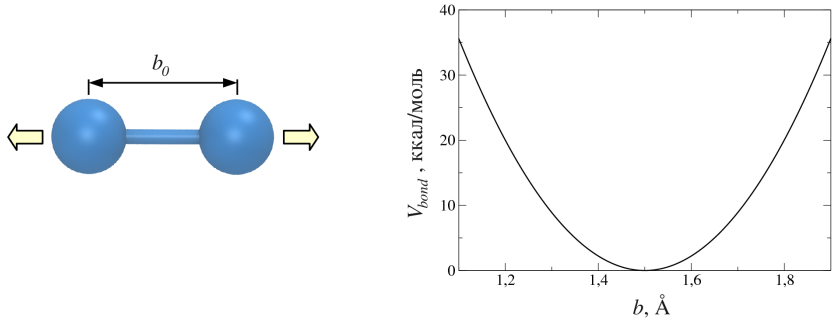


Рис. 25: Вибрации ковалентной связи (слева) и общий вид гармонического потенциала (справа).

Этот потенциал сохраняет длину ковалентной связи. Наиболее общая форма ковалентного потенциала V_{bond} основывается на законе Гука (гармонический потенциал). В этом случае:

$$V_{bond} = \frac{1}{2} K_s (b - b_0)^2, \tag{6}$$

где b - расстояние между двумя ковалентно связанными атомами, b_0 - его равновесное значение, а K_s - коэффициент упругости. Значение равновесного расстояния b_0 может быть получено из рентгеновской кристаллографии или с использованием вычислений из начальных принципов (*ab initio* вычисления). Коэффициенты упругости можно получить из результатов ИК-поглощения и спектроскопии комбинационного рассеяния (сокр. КР, иначе рамановская спектроскопия, англ. Raman scattering spectroscopy). Частота флуктуаций длины ковалентной связи при этом равна $\omega^2 = \frac{K_s}{m}$, где m - масса атома. Если величины ковалентно связанных атомов сравнимы (в таком случае необходимо учитывать массы обоих частиц), то должна использоваться эффективная масса $\mu = m_1 m_2/(m_1+m_2)$. В таких силовых полях как CHARMM и OPLS используется гармоническое приближение (Уравнение 6). Параметры K_s и b_0 для всех возможных пар типов атомов, формирующих ковалентную связь, при вычислениях берутся из файла параметров силового поля (для CHARMM27 файл параметров называется “par_all27_prot_na.prm”, который поставляется и должен использоваться в паре с соответствующим файлом топологий – “top_all27_prot_na.rtf”). Короткая выборка из фрагмента файла параметров, описывающая потенциал ковалентных связей, выглядит следующим образом:

```
1 BONDS
2 !Types     Ks          b0       Comment
3 CT1  C     250.000     1.4900 ! Backbone CA-C bond
4 CT1  CT1   222.500     1.5000 ! CA-CB bond, e.g. in Valine
5 NH1  H     440.000     0.9970 ! Backbone N-H bond
6 OT   HT    450.000     0.9572 ! O-H bond in water
```

Для каждой пары типов атомов приведены коэффициент упругости K_s в ккал/(моль·Å^2) и равновесное значение для длины связи b_0 в Å. В комментарии обычно указывается тип связи или источник данных параметров (ссылка на научную статью или описание метода получения параметров).

Использование гармонического приближения обосновано только при небольших (<10%) вибрациях длины ковалентной связи. Хотя при больших отклонениях от первоначальной

длины ковалентная связь разрушается, использование гармонического приближения этот эффект описать не может. При необходимости учитывать возможный разрыв связи, можно использовать потенциал Морза, который задаётся следующим уравнением:

$$V_{bond}^{M} = D(1 - e^{-S(b-b_0)})^2, \tag{7}$$

где D и S определяют значение потенциальной энергии при $b \to \infty$ (энергия диссоциации связи) и ширину потенциальной ямы. Хотя потенциал Морза может описать разрыв ковалентной связи, его значение при $b \to 0$ конечно (Рис. 26). Так как при малых отклонениях b от b_0 потенциал Морза можно приблизить гармоническим потенциалом, параметры D и S можно связать с параметрами в уравнении 6. Использование потенциала Морза позволяет более точно описать вибрации ковалентных связей. В качестве промежуточного варианта можно использовать несколько дополнительных членов разложения V_{bond}^{M} (Уравнение 7), добавив их в уравнение 6 (например, кубический член или член четвёртой степени). На практике вибрации ковалентных связей лёгких атомов (атомов водорода) могут не учитываться, а длина этих связей может быть зафиксирована при помощи таких алгоритмов как SHAKE или RATTLE.

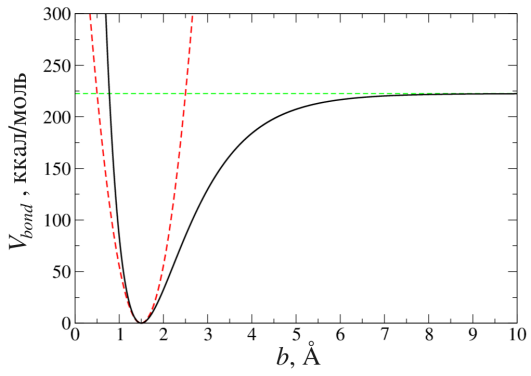


Рис. 26: Потенциал Морза. Красным пунктиром показано гармоническое приближение, зелёным – значение энергии диссоциации.

6.2.2 Вибрации углов

Потенциал вибрации углов используется для того, чтобы сохранить геометрию ковалентных связей в молекуле. В реальной молекуле эта геометрия которая контролируется гибридизацией орбиталей электронов атомов. Например, гибридизация sp позволяет формирование двух связей, расположенных под углом 180°. Для других типов гибридизации, таких как sp2 и sp3, соответствующие равновесные значения углов равны 120° и $109,5°$. При параметризации потенциалов вибрации углов необходимо также учитывать влияние локального окружения молекулы. Например, для молекулы воды, в которой гибридизация атома кислорода имеет тип sp3, равновесное значение угла между двумя OH-связями составляет 105°.

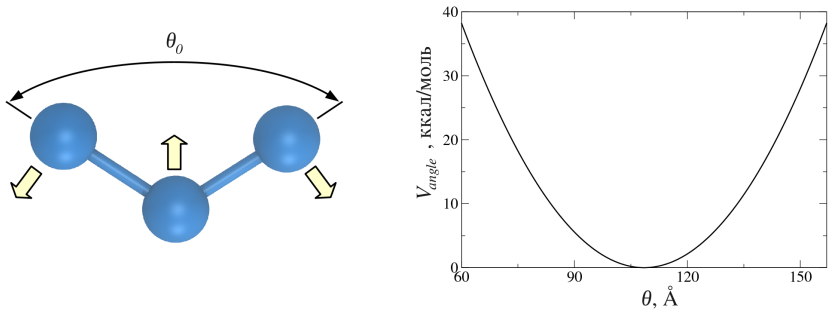


Рис. 27: Вибрации углов между двумя ковалентными связями (слева) и общий вид гармонического потенциала (справа).

В молекулярной механике используется два типа потенциала вибрации углов – гармонический потенциал:

$$V_{angle} = K_\theta(\theta - \theta_0)^2 \tag{8}$$

и тригонометрический потенциал:

$$V^T_{angle} = K^T_\theta(\cos\theta - \cos\theta_0)^2. \tag{9}$$

Очевидно, что при $\theta \to \theta_0$ $V^T_{angle} = K^T_\theta(\theta - \theta_0)^2 \sin^2\theta_0$ и потенциалы совпадают. Большинство силовых полей используют гармоническую форму потенциала вибрации углов

(Уравнение 8), хотя вычисление силы для тригонометрической формы (Уравнение 9) более эффективно (при расчёте производной нет необходимости считать обратные тригонометрические функции). Параметры гармонического потенциала для описания вибраций углов между ковалентными связями в файле параметров силового поля CHARMM27 ("par_all27_prot_na.prm") перечислены в секции "ANGLES":

```
ANGLES
!atom types      Ktheta   Theta0   [Kub    S0] ! Comment
NH1  CT1  C      50.000   107.0000 ! N-CA-C angle in backbone
HT   OT   HT     55.000   104.5200 ! H-O-H angle in water
CT3  CT1  CT1    53.350   108.50    8.00   2.56100 ! Angle with
                                    ! Urey-Bradley parameters (e.g.
                                    ! CA-CB-CG angle in Valine)
```

Первые три столбца идентифицируют типы атомов, в третьем и четвёртом столбце приведены коэффициент упругости K_θ в ккал/моль и равновесное значение угла θ_0 в градусах. В некоторых силовых полях для некоторых из углов (например в CHARMM27 для угла CT3-CT1-CT1) вводятся дополнительные корректировки Урея-Брэдли, которые представляют собой обычный гармонический потенциал, связывающий два атома, разделённых двумя гармоническими связями. Для задания данного потенциала используется постоянная упругости K_{UB} (в ккал/(моль·Å^2) и равновесное расстояние S_0 (в Å). Для большинства углов параметры для поправки Урея-Бредли не задаются, и поправка к потенциалу не считается. Логично, что значение коэффициента упругости K_{UB} значительно меньше коэффициента упругости K_s ковалентных связей.

6.2.3 Торсионные углы

Из-за относительно высоких энергетических параметров вибраций ковалентных связей и углов, колебания данных степеней свободы при комнатной температуре невелики. Это оправдывает применение гармонического приближения потенциальной функции. Энергетические метрики остальных компонентов функции потенциальной энергии (Уравнения 3-

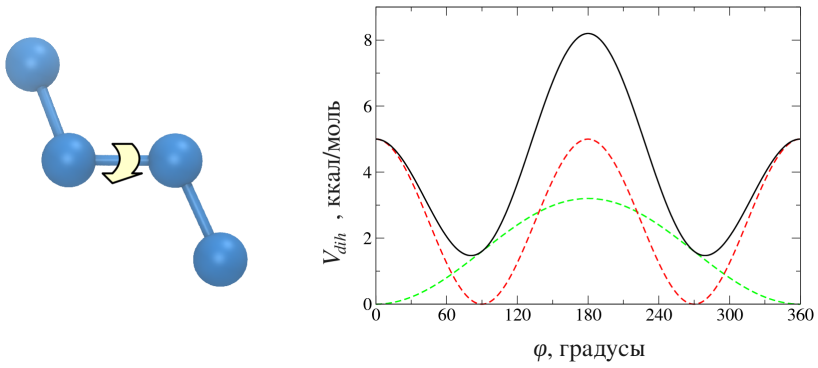


Рис. 28: Вибрации торсионных углов (слева) и общий вид потенциала (справа). Пунктирными линиями на графике показаны два слагаемых потенциала угла с разной периодичностью. Сплошная линия является суммой этих слагаемых и описывает общий вид потенциала торсионного угла.

5) сопоставимы с энергией температурных флуктуаций k_BT. В результате, именно эти компоненты определяют структурные преобразования в биомолекулах. Потенциал торсионных углов играет важную роль в формировании локальных конформаций, его необходимость связана с тем фактом, что потенциалы ковалентных связей и углов, а также потенциалы невалентных взаимодействий, не могут точно описать энергетику углеводородных молекул. Например, углеводородная цепь бутана (C_4H_{10}) имеет минимальную энергию в так называемой цис-конформации, в которой все 4 атома углерода находятся в одной плоскости (Рис. 29). Природа потенциала торсионных углов может быть объяснена наложением орбиталей электронов ковалентных связей. Общая функциональная форма потенциала задаётся следующим уравнением:

$$V_{dih} = \sum_{n=n_1,\ldots,n_p} K_\phi(1 - \cos(n\phi - \phi_0)), \tag{10}$$

где K_ϕ – высота барьера свободной энергии, n - целое число, определяющее периодичность потенциала (то есть количество минимумов в интервале $[0, 2\pi]$), а ϕ_0 показывает положение центрального минимума. В описании белковых молекул используются значения $n = 1$, 2, 3 или 4, наиболее часто встречающиеся – 1 и 2. Кроме того, одному торсионному углу может соответствовать несколько наборов параметров с разной периодичностью ($n = n_1, \ldots, n_p$).

В этом случае, потенциалы с разной периодичностью складываются. Пример такого потенциала показан на рисунке 28. Выборка параметров торсионных углов из силового поля

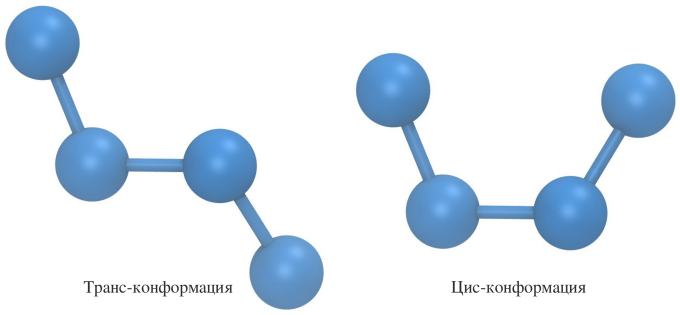


Рис. 29: Торсионные углы в цис- и транс- конформации.

CHARMM27 выглядит следующим образом:

```
DIHEDRALS
!atom types             Kphi    n   phi0   ! Comment
C    CT1  NH1  C        0.2000  1   180.00 ! Backbone phi
NH1  C    CT1  NH1      0.6000  1     0.00 ! Backbone psi
CT2  C    NH1  CT1      1.6000  1     0.00 ! Angle with two
CT2  C    NH1  CT1      2.5000  2   180.00 ! different periodicities
```

Первые два угла ($n = 1$) имеют минимум энергии при $\phi = 0$ (цис-конформация) или при $\phi = 180°$ (транс-конформация). Следующий набор параметров ($n = 2$) задаёт потенциал с двумя минимумами при $\phi = 0°$ и 180°. При $n = 3$ у потенциала три минимума при $\phi = 60°$, 180° и 300°. В случаях, когда потенциал имеет более одного минимума, их глубина совпадает, а для того, чтобы получить потенциал с несколькими минимумами разной глубины, можно использовать сумму потенциалов с разными параметрами и разным количеством минимумов. Примером такого угла может служить угол между атомами типов CT2, C, NH1 и CT1. Параметры для потенциала торсионных углов обычно получают при помощи вычислений на основе квантовой механики (вычисления “из начальных принципов” – *a*b initio), используется также дополнительная оптимизация параметров, основанная на результатах экспериментов.

6.2.4 Неправильные торсионные углы

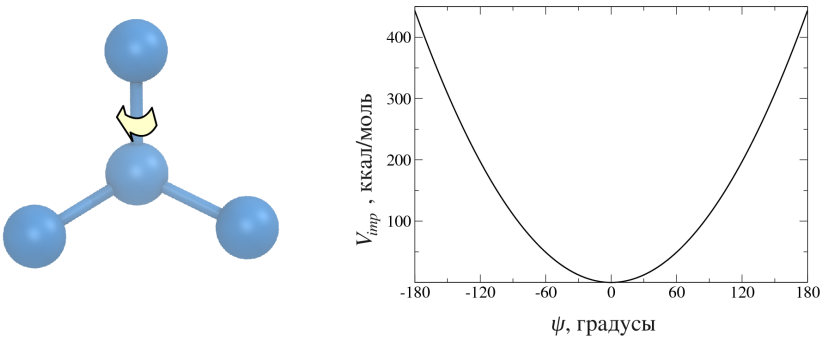


Рис. 30: Схематическое изображение компоновки атомов в неправильном торсионном угле. Справа показан вид функции потенциальной энергии.

Неправильные торсионные углы (Рис. 30) используются для описания хиральности (правильной зеркальной компоновки) атомов. Рассмотрим атомы j, k и l, ковалентно связанные с центральным атомом i. Неправильный торсионный угол при этом определяется как угол между плоскостью (ijk) и вектором (il). Обычно, потенциал неправильного торсионного угла задаётся формулой:

$$V_{imp} = K_\psi(\psi - \psi_0)^2, \tag{11}$$

где коэффициент K_ψ определяет жёсткость угла, а ψ_0 – его равновесное значение. Параметры потенциала неправильных торсионных углов также можно найти в файле параметров силового поля:

```
IMPROPER
!atom types             Kpsi          -       psi0    ! Comment
O     CT1   NH2   CC      45.0000          0       0.0000 !
O     X     X     C      120.0000          0       0.0000 !
```

Как и в случае правильных торсионных углов, первыми в строчке перечисляются типы атомов. Затем – коэффициент упругости K_ψ, периодичность (не используется в данном потенциале) и равновесный угол ψ_0. В первом наборе параметров все типы атомов O, CT1,

NH2 и CC точно определены, в то время как во втором наборе присутствуют неопределённые типы атомов X, которые могут быть заменены на любой тип атома. Параметры для типов O, X, X и C могут быть использованы, например, для угла между плоскостью C_α-C-N и связью C-O в пептидном скелете белка. В этом случае, два неопределённых типа атомов X соответствуют атомам C и N. Для конкретного набора типов атомов сначала ищется набор параметров с точным совпадением типов и лишь потом – подходящий набор параметров с неопределёнными типами. Таким образом, параметры с явно заданными типами имеют приоритет. Конечно, подобный потенциал можно задать и при помощи обычных торсионных углов, но, так как планарность ($\psi = 0°$) угла должна быть фиксирована ($K_\psi = 120$ ккалл/(моль·градус^2)), менее вычислительно затратно использовать неправильный вид потенциала.

6.2.5 Невалентные взаимодействия

Невалентные взаимодействия включают в себя силы Ва-дер-Ваальса и электростатические взаимодействия. Силы Ван-дер-Ваальса описывают сильное отталкивание атомов, находящихся близко друг к другу и слабое притяжение на большом расстоянии. Потенциал этого взаимодействия для двух атомов i и j обычно имеет форму потенциала Леннарда-Джонса:

$$V_{LJ} = \frac{B_{ij}}{r_{ij}^{12}} - \frac{A_{ij}}{r_{ij}^{6}}, \tag{12}$$

где B_{ij} и A_{ij} – коэффициенты, определяющие положение минимума энергии и его глубину. Правила, по которым из приведённых параметров можно рассчитать коэффициенты B_{ij} и A_{ij}, задаются следующим образом:

$$\begin{aligned} A_{ij} &= 2\varepsilon_{ij}\sigma_{ij}^6; \\ B_{ij} &= \varepsilon_{ij}\sigma_{ij}^{12}; \\ \varepsilon_{ij} &= \sqrt{\varepsilon_i \varepsilon_j}; \\ \sigma_{ij} &= \frac{1}{2}\left(\sigma_i + \sigma_j\right). \end{aligned} \tag{13}$$

Параметры ε_i и σ_i задаются в файле параметров силового поля в секции "NONBONDED":

```
NONBONDED
!atom    -    epsilon   Rmin/2 [ -    eps,1-4   Rmin/2,1-4] ! Comment
C       0.0   -0.1100   2.0000 ! Backbone C
CA      0.0   -0.0700   1.9924 ! C-alpha atom
H       0.0   -0.0460   0.2245 ! Polar H
CT1     0.0   -0.0200   2.2750   0.0   -0.01000 1.9000       ! Aliphatic sp3 C
```

Первый столбец идентифицирует тип атома. Второй и пятый столбец игнорируются. Третий и четвёртый столбцы содержат ε_i в ккалл/моль и σ_i в Å. Для некоторых типов атомов (например для CT1) также задаются изменённые параметры ε_i и σ_i, которые используются, когда рассматриваемые атомы разделены тремя ковалентными связями.

Отталкивание атомов, расположенных близко друг к другу, связано с принципом исключения Паули, а слабое притяжение удалённых атомов – силами дисперсии Лондона. Параметры ε_i и σ_i можно получить из результатов экспериментов рентгеновской кристаллографии и из моделирования небольших органических молекул. Важно обратить внимание на то, что при $r \to \infty$ потенциал V_{LJ} быстро убывает ($\sim r^{-6}$). Поэтому, взаимодействия Ван-дер-Ваальса называют короткодействующими. Несмотря на то, что возможное количество пар $i-j$ в системе из N атомов равно N^2, существуют эффективные методы сократить время расчёта этого потенциала.

Электростатический потенциал или потенциал Кулона описывает взаимодействия между заряженными атомами. Энергия потенциала Кулона может быть записана как:

$$V_{el} = \frac{q_i q_j}{4\pi\varepsilon_0 \varepsilon(r_{ij}) r_{ij}}, \tag{14}$$

где q_i и q_j – частичные заряды атомов i и j, $\varepsilon(r_{ij})$ – зависящая от расстояния функция диэлектрической проницаемости среды, а ε_0 – диэлектрическая постоянная. Если водное окружение молекулы представлено явно в виде молекул воды, то диэлектрическая проницаемость среды автоматически будет принята во внимание. В этом случае $\varepsilon(r_{ij}) = 1$. Необходимость учитывать зависимость ε от r возникает при использовании неявно задан-

ного растворителя, так как в воде $\varepsilon = 80$.

7 Молекулярная динамика

7.1 Идея молекулярной динамики

Описанное выше силовое поле, которое опирается на принципы молекулярной механики (потенциальную функцию), может быть использовано для разных целей (для методов Монте-Карло, поиска минимальной энергии, докинга, динамики Ланжевена и т.д.). Другими словами, молекулярная механика не ограничивает использование силового поля. В случае молекулярной динамики, использование выведенной из принципов молекулярной механики потенциальной функции силового поля, позволяет вычислить значения сил, действующих на все атомы системы. Эти силы затем могут быть использованы при численном интегрировании уравнений движения.

Молекулярная динамика (МД) – метод вычисления равновесных и кинетических свойств системы, которые подчиняются классическим законам физики. Любой процесс с характеристическим временем $\tau > \tau_q \sim 2$пс может быть описан при помощи классической физики. Поэтому, за исключением вибраций ковалентных связей, все движения в молекулярной динамике можно описывать при помощи классических законов движения. Основным преимуществом молекулярной динамики является её способность получить “реальную” микроскопическую динамику, подчиняющуюся ландшафту свободной энергии системы и межатомным взаимодействиям. Программная реализация молекулярной динамики включает в себя следующие шаги:

1. Задание параметров, описывающих условия молекулярных симуляций, таких как температура, количество атомов и так далее.

2. Инициализация, которая включает в себя задание координат атомов и распределение начальных скоростей атомов.

3. Вычисление сил.

4. Численное интегрирование уравнений движения.

5. Повтор шагов 3 и 4, пока не будет достигнут желаемый временной интервал.

6. Вычисление средних значений.

7.2 Начальная конформация

После того, как параметры молекулярной динамики установлены, читается начальная структура. Важно, чтобы данная конформация не включала в себя наложения атомов друг на друга, необычные локальные конформации, что может повлечь за собой слишком большие значения сил и нестабильность численного интегрирования.

7.3 Начальное распределение скоростей

Начальное распределение скоростей атомов может быть получено при помощи распределения Максвелла-Больцмана:

$$p(\upsilon_{i,\alpha}) = \left(\frac{m}{2\pi k_B T}\right)^{\frac{1}{2}} \exp\left[-\frac{m\upsilon_{i,\alpha}^2}{2k_B T}\right], \tag{15}$$

где $\upsilon_{i,\alpha}$ – компонента ($\alpha = x, y, z$) вектора скорости для атома i. Это распределение может быть использовано для получения температуры системы используя соотношение:

$$\langle\frac{m\upsilon_{i,\alpha}^2}{2}\rangle = \frac{1}{2}k_B T. \tag{16}$$

7.4 Вычисление атомарных сил

Потенциальная функция силового поля позволяет рассчитать энергию системы. Как известно, производная этой энергии по координате атома будет равна силе, действующей

на этот атом, что в векторном виде можно записать следующим образом:

$$\vec{f_i} = -\begin{pmatrix} \frac{\partial}{\partial x_i} \\ \frac{\partial}{\partial y_i} \\ \frac{\partial}{\partial z_i} \end{pmatrix} V. \tag{17}$$

Как и полная энергия системы, сила, действующая на атом, может быть разложена в сумму сил, каждая из которых будет соответствовать одному из потенциалов. В молекулярной динамике используются явные формулы для расчёта потенциала – численное интегрирование потенциала потребовало бы слишком большого числа вычислений, так как функция потенциальной энергии зависит от $3N$ переменных. Поэтому, перед реализацией того или иного потенциала следует вычислить его производную и строить вычислительные процедуры уже для производной потенциала.

7.5 Интегрирование уравнений движения

Рассмотрим уравнение движения Ньютона:

$$m\frac{d^2\vec{r}_i}{dt^2} = \vec{f_i} \tag{18}$$

Используя разложение Тейлора, уравнение 18 и формулу $r' = \upsilon$, для координаты атома r можно записать:

$$\vec{r}_i(t+\Delta t) = \vec{r}_i(t) + \vec{v} - i(t)\Delta t + \frac{\vec{f}_i(t)}{2m}\Delta t^2 + \frac{1}{6}\frac{d^3\vec{r}_i(t)}{dt^3}\Delta t^3 + \vec{o}(\Delta t^4) \tag{19}$$

$$\vec{r}_i(t-\Delta t) = \vec{r}_i(t) - \vec{v}_i(t)\Delta t + \frac{\vec{f}_i(t)}{2m}\Delta t^2 - \frac{1}{6}\frac{d^3\vec{r}_i(t)}{dt^3}6\Delta t^3 + \vec{o}(\Delta t^4) \tag{20}$$

Сложив эти уравнения, получим:

$$\vec{r}_i(t+\Delta t) = 2\vec{r}_i(t) - \vec{r}_i(t-\Delta t) + \frac{\vec{f}_i(t)}{m}\Delta t^2 + \vec{o}(\Delta t^4). \tag{21}$$

Скорости, необходимые для расчёта кинетической энергии и температуры, могут быть найдены при помощи центральной разностной схемы:

$$\vec{v}_i(t) = \frac{\vec{r}_i(t + \Delta t) - \vec{r}_i(t - \Delta t)}{2\Delta t} + \vec{o}(\Delta t^2). \tag{22}$$

Уравнения 21 и 22 описывают один из самых популярных в молекулярной динамике алгоритмов интегрирования уравнений движения – алгоритм Верле. Стоит обратить внимание на то, что точность в уравнении для расчёта смещения координат (Ур. 21) больше, чем в уравнении для скоростей (Ур. 22). При интегрировании используются силы, рассчитанные на данном шаге по времени. Так как силы являются функцией координат атомов, то после смещения координат и изменения скоростей при численном интегрировании, силы нужно рассчитывать заново. Этот процесс является основой молекулярной динамики.

7.6 Периодические граничные условия

Зачастую в молекулярном моделировании необходимо ограничить объём системы. Это происходит, например, при моделировании в явном растворителе – когда в системе присутствуют молекулы воды. Введение отталкивающих граничных условий не всегда оправдано, так как в этом случае будут существенные граничные эффекты. Обычно из положения выходят, вводя периодические граничные условия, основная идея которых – замкнуть систему через противоположные границы (Рис. 31). При этом, атом пересекающий границу, влетает в систему с противоположной стороны. При расчёте расстояний в условиях периодических граничных условий необходимо учитывать замкнутость системы. Поэтому, расстояние между атомами считается как:

$$\vec{r}_{ij} = \vec{r}_j - \vec{r}_i - [(\vec{r}_j - \vec{r}_i)/L] \cdot L, \tag{23}$$

где квадратные скобки обозначают округление. Для большинства потенциалов изменение формулы расчёта расстояний является единственным эффектом введения периодических граничных условий. Проблемы возникают при расчёте электростатического взаимодей-

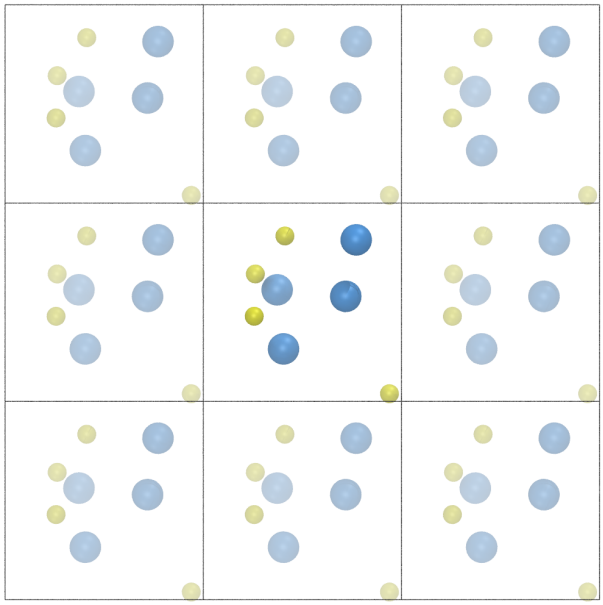


Рис. 31: Схематическое изображение использования периодических граничных условий в молекулярной динамике. Атомы, покинувшие основную ячейку (в центре), входят в неё с противоположной стороны. При расчёте потенциалов, необходимо учитывать наличие “мнимых ячеек”, т.е. расчёт расстояний должен производится с учётом замыкания ячейки по граням.

ствия: так как электростатический потенциал убывает в зависимости от расстояния как $\sim 1/r$, сумма вкладов бесконечного числа периодических образов исходной ячейки не будет сходится. Поэтому при расчёте электростатических взаимодействий в условиях периодических граничных условий используют методы суммирования Эвальда на сетке (методы PME, SPME и PPME [23–25]).

8 Уровни детализации молекулярного моделирования

Как правило, при моделировании биологической задачи N тел, приходится искать компромисс между уровнем детализации и скоростью расчёта. Выбор правильного уровня детализации позволяет исследователям решать конкретные задачи, возникающие для той или иной биологической системы. Для некоторых из них необходимо использовать более детальное полноатомное приближение (МД), иногда же более точные результаты можно получить при помощи крупнозернистой модели и динамики Ланжевена (ДЛ). В то время, как МД – более точный подход к моделированию, ДЛ позволяет моделировать на биологически-важных временных интервалах (мс-с). Численные процедуры, используемые в обоих методах, очень похожи. Взаимодействия между частицами (например, силы Ван-дер-Ваальса, электростатические взаимодействия, ковалентные связи) во всех случаях описываются при помощи потенциальной функции силового поля. Эта функция зависит от координат всех частиц (атомов или групп атомов), их типов, зарядов и описываемых степеней свободы (вибраций, поворотов,торсионных вращений и т.д.). В динамике Ланжевена часто используются уникальные параметры для каждой пары взаимодействующих частиц, которые задаются, основываясь на их расположении в нативной структуре белка (так называемые нативные контакты). Используя силовое поле, рассчитываются атомарные и молекулярные силы, действующие на все частицы в системе. Затем частицы перемещаются согласно законам движения (Ньютона или Ланжевена).

Так как молекулярное моделирование основывается на эмпирической функции взаимодействия между частицами, результаты моделирования должны быть сравнены с тем или иным экспериментом. Это делается для того, чтобы убедиться в адекватности задания параметров и функций силового поля. В силу того, что временные шкалы эксперимента (мс-мин) и моделирования в полноатомном разрешении (нс-мкс) не совпадают, прямое сравнение провести практически невозможно. С ростом вычислительных возможностей компьютерной техники становится возможным увеличить время моделирования, что иногда позволяет найти новые недостатки современных силовых полей [12, 26]. Последние постоянно претерпевают изменения: появляются новые наборы параметров, изменяется

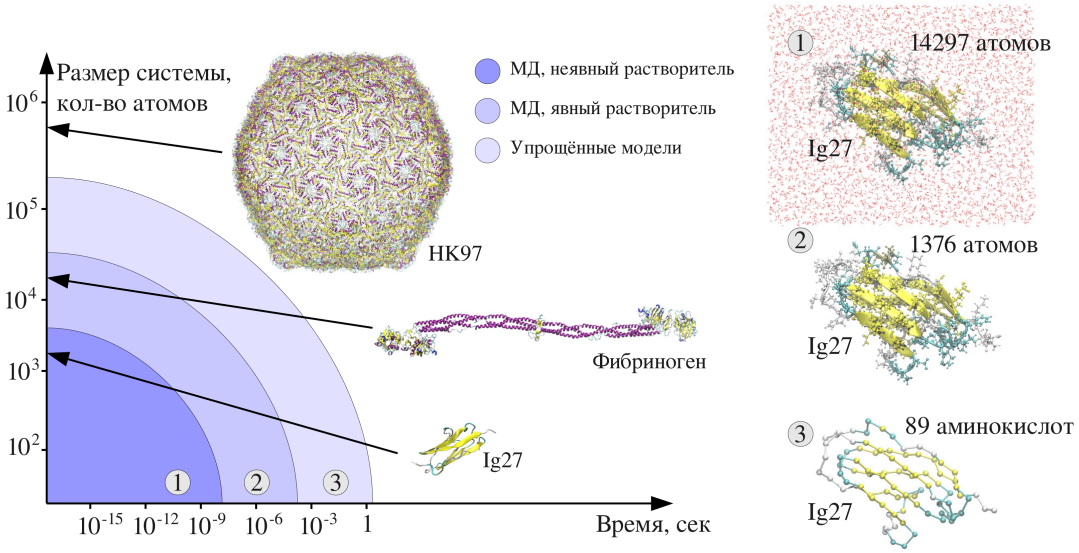


Рис. 32: Примерные временные интервалы, достигаемые на одном ЦП для различных размеров систем и разных способов моделирования. На графике изображены различные системы: домен титина $Ig27$, мономер фибриногена и капсид $HK97$, для которых стрелкой показаны примерные размеры в атомах. Справа показаны размеры систем, получаемых при моделировании в явном растворителе (сверху), в неявном растворителе (в центре) и при использовании упрощённой C_α-модели. Видно, системы размера фибриногена уже невозможно моделировать на экспериментальном временном интервале с использованием ЦП.

функциональная часть, вводится зависимость между параметрами разных частей потенциальной функции (так называемые поляризуемые силовые поля) [27]. Кроме улучшения аппаратной части, можно применять и другие способы увеличения производительности вычислений. Среди таких способов использование неявного растворителя и упрощение моделей биомолекул. При использовании упрощённых моделей число рассматриваемых степеней свободы уменьшается, так как остаются только те, которые важны для решаемой задачи. Обычно в упрощённых моделях одна аминокислота белка или одна нуклеиновая кислота ДНК/РНК рассматривается как отдельная частица или пара частиц [28, 29]. В силу того, что при этом структурные единицы системы увеличиваются в размерах, шаг по времени также может быть увеличен. Из-за того, что при применении данного подхода вычислительная точность ниже, чем в полноатомном разрешении, упрощённые модели часто

разрабатываются для моделирования конкретных свойств биомолекул. Например, модель SOP была разработана для описания механических свойств, существуют также модели для описания белок-белковых взаимодействий, фолдинга белков и так далее [28, 29]. Использование неявного растворителя также позволяет уменьшить количество степеней свободы. В данном подходе механическое воздействие воды описывается при помощи случайной силы, действующей на атомы системы, а энергетическое воздействие (гидрофобный эффект, заземление электростатических взаимодействий) описывается при помощи дополнительной потенциальной функции. При этом межатомные взаимодействия описываются также, как и когда вода представлена явно (при моделировании в явном растворителе). Применение неявного растворителя не только помогает ускорить вычислительный процесс, но также более привлекательно для некоторых задач. Например, можно явно оценить энергетическую составляющую действия растворителя на биомолекулу, быстрее менять её макросостояния.

Часть III

Молекулярная динамика с использованием пакета NAMD

Идея молекулярной динамики, конечно, не нова. За последние десятилетия было разработано несколько программных пакетов, которые позволяют производить молекулярное моделирование. Некоторые из них – бесплатные с открытым исходным кодом (такие как например пакеты NAMD [15] и GROMACS [16]), существуют также коммерческие программные продукты – CHARMM [14], AMBER [17]. Цена последних для некоммерческих организаций, таких как ВУЗы, невысока, поэтому многие исследовательские группы предпочитают использовать именно эти программные пакеты. В данной части мы остановимся на пакете NAMD и рассмотрим по шагам, как данный программный продукт может быть использован для проведения моделирования простой белковой системы в явном растворителе. В силу того, что начальная конформация биомолекул получается экспериментально или с использованием численных алгоритмов, проведение моделирования в физиологических условиях требует нескольких предварительных вспомогательных шагов, часто называемых приготовлением системы (Рис. 33).

Далее мы более детально остановимся на каждом из перечисленных этапов моделирования.

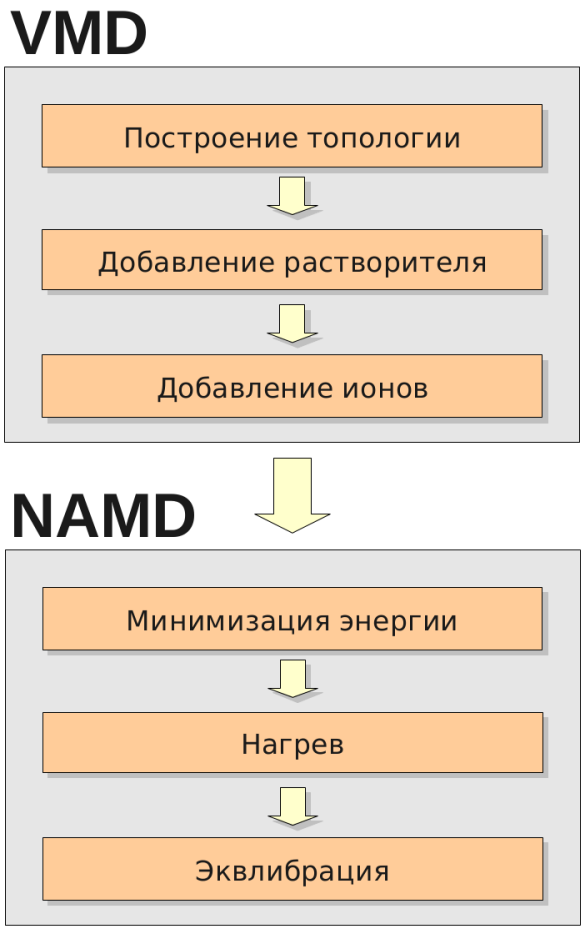


Рис. 33: Предварительные (подготовительные) этапы моделирования. Программа VMD используется для построения топологии, добавления растворителя и ионов. В NAMD, на начальных этапах, производится минимизация энергии, нагрев и эквилибрация системы. По завершению этих этапов можно приступать к непосредственному моделированию с использованием пакета NAMD.

9 Начальные данные

Для проведения моделирования при помощи пакета молекулярной динамики потребуются следующие начальные данные:

1. Файл координат (.pdb). Данный файл содержит координаты атомов в декартовой системе координат.

2. Файл топологии (.psf). В данном файле содержится описание всех атомов в системе, включая их типы, заряды, массы, а также информация о ковалентных взаимодействиях: пары атомов, связанных ковалентно, тройки атомов, образующих углы, четвёрки атомов, образующих торсионные углы.

3. Файл параметров силового поля (например, par_all22_prot.inp). В данном файле содержатся параметры для потенциалов силового поля – постоянные упругости ковалентных связей, равновесные расстояния, углы и т.д.

В качестве начальных координат атомов чаще всего выступают данные экспериментальных исследований, таких как рентгеновская кристаллография или эксперимент ЯМР (ядерный магнитный резонанс). Большинство научных проектов, связанных с молекулярным моделированием, начинается с поиска подходящих трёхмерных структур на таких интернет-ресурсах, как база данных белковых структур (Protein Data Bank[2]). На данном ресурсе собрано огромное количество результатов экспериментов со ссылками на оригинальные статьи исследователей, проводивших эксперимент. Стоит отметить, что особенности экспериментальных процедур не всегда позволяют точно определить местоположение всех атомов в системе, а положение водородов в рентгеновской кристаллографии вообще не определяется. Поэтому, перед тем как начать процесс моделирования той или иной биологической молекулы, необходимо тщательно изучить полученную экспериментаторами структуру.

[2]http://www.pdb.org

Вторым шагом является создание файла топологии системы (для пакета NAMD этот файл имеет расширение .psf). В файле топологии содержится информация об атомах, входящих в систему, а также о связанности этих атомов в белке. Для того, чтобы получить данный файл, необходимо использовать программу VMD [3], создаваемую той же научной группой, которая создала программный пакет NAMD. В самых простых случаях файл топологии можно построить при помощи автоматического построения, через графический интерфейс программы VMD. При этом будет создано два файла – .psf и обновлённый .pdb. Последний будет содержать неразрешённые в экспериментах атомы водорода и небольшие неразрешённые участки полипептидной цепи, если такие имеются.

10 Программный пакет VMD и подготовка к моделированию

Программный пакет VMD предлагает удобный интерфейс для визуализации биомолекул, построения биомолекулярных структур и первичной обработки результатов моделирования. Также этот пакет тесно интегрирован с программой NAMD и позволяет подготавливать все необходимые этой программе данные. VMD распространяется бесплатно в виде открытого кода и может быть загружен с сайта университета Иллинойса США[3]. На сайте также можно найти большое количество документации по пакету для самостоятельного и более глубокого его изучения. В рамках данного курса мы остановимся лишь на основной функциональности пакета VMD.

10.1 Файл координат .pdb

Для того, чтобы начать работу с биомолекулой, необходимо получить её трёхмерную структуру. Один из самых популярных источников – база данных белковых структур[4]. Сама база данных была основана в 1971 году в Брукхейвенской национальной лаборатории. Тогда в этой базе данных содержалось всего 7 структур, одной из которых стала структура белка миоглобин, разрешённая ещё в 1958 году группой Джона Кендрю (John Kendrew) [30]. Для изучения трёхмерной структуры миоглобина, Джону Кендрю и его группе пришлось вручную создавать модель молекулы (в статье приведены её фотографии). Для обмена результатами эксперимента в те времена использовались перфокарты. Так как каждый атом разрешённой структуры хранился на отдельной перфокарте, передача результатов была довольно трудоёмким процессом – координаты миоглобина занимали порядка 1000 перфокарт. Централизованное хранилище данных помогло не только существенно упростить такой обмен, но и сделать разрешённые структуры более доступными. В 80-х годах, вместе с возможностями эксперимента, возрос и интерес к этой области. Количество структур

[3] http://www.ks.uiuc.edu/Research/vmd/
[4] http://www.pdb.org

начало расти экспоненциально. Вскоре технический прогресс позволил упростить и способ обмена разрешёнными структурами, а с появлением сети интернет и персональных компьютеров – результаты работы кристаллографов стали доступны каждому. На сегодняшний день база данных белковых структур содержит порядка 80 000 структур различных биомолекул. Наиболее популярным форматом файла координат атомов биомолекулярной системы до сих пор остаётся формат .pdb, в котором каждому атому системы отведена одна строчка. Формат имеет некоторые ограничения, сохранившиеся ещё со времён перфокарт. В частности, длина каждой строки файла не может превышать 80 символов, каждому из которых отведена своя роль. Так как основной целью формата .pdb было обеспечить возможность обмениваться результатами экспериментальных исследований, он подразумевает наличие большого числа информации об использованной в эксперименте процедуре, количестве неразрешённых атомов и так далее. Нас главным образом будет интересовать описание атомов системы, разбиение которого по столбцам приведено в таблице 1.

Таблица 1: Описание формата .pdb

Столбец	Тип	Содержит
1-6	Фикс. выражение	Строка “ATOM␣␣”
7-11	Целое	Порядковый номер атома
13-16	Строка (4 символа)	Название атома
17	Символ	Индикатор неточности позиционирования атома (если в файле есть дубликат с другими координатами)
18-20	Строка (3 символа)	Имя аминокислоты
22	Символ	Идентификатор полипептидной цепи
23-26	Целое	Порядковый номер аминокислоты
27	Символ	Код добавления аминокислот
31-38	Число с пл. тчк.(8.3)	Координата X атома в декартовой системе (в ангстремах)
39-46	Число с пл. тчк.(8.3)	Координата Y атома в декартовой системе (в ангстремах)
47-54	Число с пл. тчк.(8.3)	Координата Z атома в декартовой системе (в ангстремах)
55-60	Число с пл. тчк.(6.2)	Заполненность (в кристаллографическом эксперименте)
61-66	Число с пл. тчк.(6.2)	Температурный множитель
73-76	Строка (4 символа)	Название сегмента молекулы (выравнивание слева)
77-78	Строка (2 символа)	Имя химического элемента (выравнивание справа)
79-80	Целое	Заряд атома

В качестве примера, рассмотрим структуру одного из доменов титина с индексом в базе данных белковых структур 1TIT. Строка описания первого атома выглядит следующим образом:

```
1.......10........20........30........40........50........60........70........80
ATOM␣␣␣␣␣␣1␣␣N␣␣␣LEU␣A␣␣␣1␣␣␣␣␣-19.761␣␣␣9.730␣␣-6.893␣␣1.00␣␣1.53␣␣␣␣␣␣␣␣␣␣␣N␣␣
```

То есть это – атом азота из аминокислоты лейцин, которая является первой аминокислотой в данном белке. Несмотря на то, что в .pdb-файле содержится детальная информация о структуре белка, представить себе эту структуру, очевидно, невозможно. Пионеры кристаллографии собирали трёхмерный модели белков [30], что, к счастью, в наши дни совсем не обязательно: существует целый ряд программных пакетов, которые позволяют визуализировать структуру белка на компьютере.

10.2 Просмотр трёхмерного изображения молекулы

В программе VMD три основных окна – окно терминала, окно изображения трёхмерной структуры и главное окно (Рис. 34). В окне терминала показывается информация о работе программы, кроме того, в нём можно напрямую писать команды при помощи языка сценариев TCL. В окне трёхмерного изображения пользователь имеет возможность поворачивать, двигать и приближать структуру молекулы. Управление программой можно осуществлять при помощи графического интерфейса главного окна. Например, чтобы загрузить структуру домена титина, необходимо через меню “File” добавить молекулу, либо указав путь к загруженному файлу .pdb либо напрямую через интернет, написав в окне четырёхсимвольный PDB-код молекулы (1TIT).

После загрузки структуры в окне изображения трёхмерной структуры отобразится молекула в виде линий ковалентных связей. При этом, ковалентные связи отобразятся основываясь как на информации из файла .pdb, где они могут быть заданы явно, так и на основе радиуса обрезки (то есть если атомы находятся достаточно близко, VMD связывает их ковалентно). Для того, чтобы изменить представление молекулы, в главном окне есть меню “Display” и пункт “Representations”. Чтобы показать элементы вторичной структуры, можно использовать метод обрисовки “New cartoons” и выделить элементы вторичной структуры цветом (“Secondary structure” в выпадающем списке “Coloring method”) Количество репрезентаций неограниченно: можно нарисовать часть молекулы и другим методом.

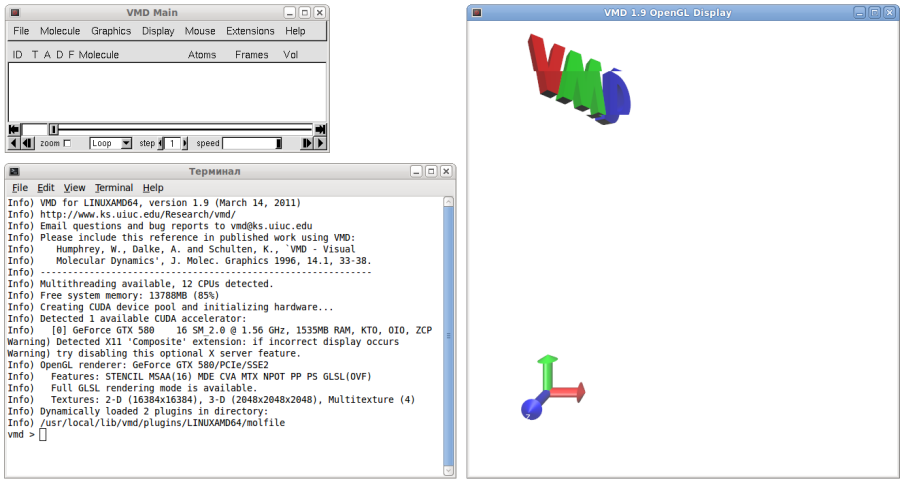


Рис. 34: Три основных окна программы VMD: главное окно, окно терминала и показа трёхмерного изображения.

Например, чтобы отобразить приведённый выше атом азота, необходимо нажать кнопку “Create rep”, в строке выбора “Selected atoms” набрать “resid 1 and name N” и установить метод обрисовки “VdW”, который отобразит этот атом в виде сферы.

Для того, чтобы переключаться между режимами работы мыши, можно использовать меню “Mouse” либо горячие кнопки на клавиатуре. Например, можно отобразить информацию об атоме (клавиша 1) или расстояние между двумя атомами (клавиша 2). Более детальную информацию о выделенных атомах или расстоянии можно узнать в пункте “Labels” меню “Graphics”. Там же можно построить график динамики наблюдаемых величин в случае, когда в программу загружено сразу много структур (как, например, при визуализации траектории молекулярной динамики).

Читателю рекомендуется уделить некоторое время для того, чтобы самостоятельно изучить основные возможности программы VMD, мы же далее остановимся на функционале этой программы, который используется для подготовки всех файлов, описывающих молекулярную систему и необходимых для проведения моделирования.

10.3 Подготовка к моделированию

10.3.1 Начальные координаты (файл PDB)

Хотя обычно моделирование системы начинается с поиска её трёхмерной структуры в базах данных белковых структур (например www.pdb.org), здесь мы рассмотрим более простой пример, в котором временные интервалы происходящих процессов позволяют получить результат даже на обычном персональном компьютере. Поэтому мы синтезируем структуру, состоящую из восьми последовательно соединённых аланинов. Известно, что аланин является одной из аминокислот, наиболее часто встречающейся в α-спиралях, процесс образования которых достаточно быстрый. Для того, чтобы синтезировать структуру, используем расширение Molefacture программы VMD. В этой подпрограмме выберем утилиту построения белков – Protein Builder, где добавим 8 аланинов в вытянутой конформации (Straight, торсионные углы: $\phi = -180°$ и $\psi = 180°$). Когда система будет готова, координаты атомов необходимо сохранить в формате PDB через меню Файл.

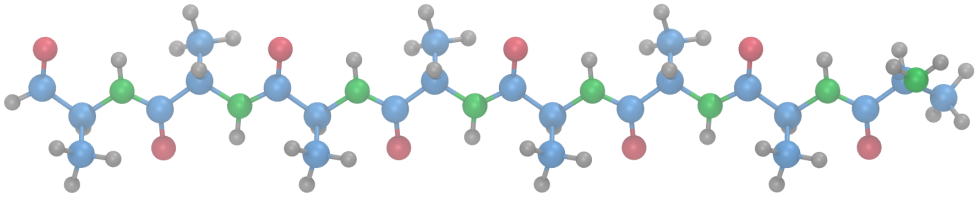


Рис. 35: Структура полиаланина в вытянутой конформации (торсионные углы $\phi = -180°$ и $\psi = 180°$). Синим показаны атомы углерода, зелёным – азота, красным – кислорода, серым – атомы водорода.

10.3.2 Создание топологии системы (файл PSF)

Файл координат не несёт в себе информации о типах атомов в системе и о её связанности. Данная информация необходима при расчёте потенциальной функции силового поля. Типы атомов, в отличие от их имён, не являются уникальными идентификаторами того или иного атома. Например, атом C_α аминокислоты аланин в силовом поле CHARMM27 будет иметь тип `CT1` – атом углерода с одним присоединённым водородом. Точно такие же

типы будут и у C_α-атомов из других аланинов в молекуле. Имя атома точно определяет местоположение этого атома внутри аминокислоты, тип атома определяет его свойства при расчёте потенциальной функции, параметры которой зависят от типов атомов вовлечённых в то или иное взаимодействие. Под связанностью системы понимают информацию о взаимодействиях внутри системы – какие взаимодействия рассчитывать для конкретных атомов в системе. Например: атом номер 1 связан с атомом номер 2 ковалентно, а атомы 1, 2 и 3 образуют ковалентный угол. При проведении моделирования эта информация используется для составления списков ковалентных связей, углов и так далее. Затем, типы атомов, участвующих во взаимодействиях, сопоставляются с типами в файле параметров силового поля. Таким образом, определяется потенциальная функция системы. Информация о типах атомах и их связанности хранится в файле структуры белка (англ. Protein Structure File, PSF).

Создание файла PSF может быть нетривиальной задачей. Особенно это относится к гетерогенным системам, таким как, например, гликопротеины. Это происходит из-за того, что в данных системах присутствуют нестандартные связи – связи между сахарами в полисахаридах и ковалентные сшивки между белками и сахаридами. В белковых системах процесс создания файла PSF значительно проще – программе необходимо передать координаты полипептидных цепей и дисульфидные связи между ними, если таковые имеются. Информацию о связанности внутри аминокислот, типы атомов и их частичные заряды при этом берутся из файла топологий используемого силового поля.

В примере с полиаланином мы будем использовать расширение программы VMD, которое предоставляет графический интерфейс для построения файла PSF. Для этого загрузим полученный ранее файл координат PDB в VMD и откроем окно расширения “Automatic PSF Builder”. Так как исследуемая система является полипептидом, настройки программы по умолчанию должны без проблем сгенерировать файл PSF. Единственное, на что следует обратить внимание – файл топологий должен соответствовать используемому нами силовому полю CHARMM27 – “`top_all27_prot_na.rtf`”.

10.3.3 Формат файла PSF

Файл PSF делится на несколько секций, каждая из которых начинается с количества элементов в данной секции и её названия. Первая секция описывает атомы системы. В нашем случае их 83, о чём свидетельствует строчка заголовка секции:

```
      83 !NATOM
        1 P1    1    ALA  N    NH3    -0.300000       14.0070           0
       ...
       83 P1    8    ALA  HB3  HA      0.090000        1.0080           0
```

Затем следуют 83 строчки, каждая из которых описывает один из атомов, для каждого из которых приведён его номер по порядку, имя сегмента молекулы (полипептидной цепи - “P1”), номер аминокислоты в этой цепи (от 1 до 8), имя аминокислоты (“ALA”), имя атома, тип атома, частичный заряд и масса в атомарных единицах.

Далее в файле PSF следует описание ковалентных связей, при этом в каждой строчке указывается до 4х связей (до 8 атомов). В следующем примере атом 1 связан ковалентно с атомом 5, атом 2 - с атомом 1, атом 1 с атомом 3, атом 4 с атомом 1 и так далее:

```
       82 !NBOND: bonds
        1       5       2       1       3       1       4       1
       ...
       80      82      80      83
```

В секциях, описывающих ковалентные углы, торсионные углы и неправильные торсионные углы атомы собраны по 3 и 4 штуки соответственно. При этом на каждую строчку приходится 3 угла (9 атомов) или 2 торсионных угла (8 атомов):

```
      147 !NTHETA: angles
        1       5       6       1       5      11       2       1       5
       ...
       81      80      83      81      80      82      82      80      83

      199 !NPHI: dihedrals
```

```
 1         5         7         8         1         5         7         9
...
79        78        80        83

15 !NIMPHI: impropers
11         5        13        12        13        11        15        14
...
76        71        78        77
```

Таким образом, файл PSF определяет, для каких атомов считать то или иное взаимодействие, а так же указывает на набор параметров, используемых при расчёте. Последнее делается через присвоение типов для всех атомов системы. Остальные секции файла находятся за рамками данного пособия.

10.4 Добавление растворителя (воды) и ионов

10.4.1 Водный раствор

Естественная среда для любой биологической системы – вода. В данном примере вода будет описываться явно, то есть в моделируемой системе будут присутствовать молекулы воды. Так как эти молекулы не связаны с пептидом и между собой ковалентно, это также потребует введения граничных условий, которые будут препятствовать разлёту молекул в пространстве. Самым популярным методом является введение периодических граничных условий.

Перед тем, как добавлять молекулы воды, необходимо оценить линейный размер системы. В конечной системе белок должен находится достаточно далеко от границ периодической ячейки. Выбор размеров ячейки очень важен. С одной стороны, чем они больше, тем больше потребуется молекул воды чтобы её заполнить и, как следствие, тем больше расчётов потребуется на каждом шаге интегрирования. С другой стороны, слишком маленькое расстояние от белка до границы с периодической ячейкой может привести к взаимодействию молекулы со своим образом в следующей ячейке. Обычно, размер ячейки

выбирается исходя из размеров молекулы l и берётся равным $L = l + 2d$, где d – отступ от границ. На выбор d влияет множество факторов: какого рода вычислительный эксперимент будет произведён, насколько сильно заряжена молекула. Например, если предполагается моделирование денатурации, то размеры молекулы будут увеличиваться и d должно быть достаточно большим, чтобы в периодическую ячейку уместилась денатурированая молекула. В случае моделирования денатурации под действием внешней силы, периодическую ячейку можно увеличить в одном направлении – совпадающем с направлением действия силы, так как маловероятно увеличение линейного размера молекулы в направлениях, перпендикулярных этому. В случае равновесных симуляций нужно учитывать возможные вращения молекулы и использовать в оценке максимальный линейный размер системы.

Выберем для полиаланина $d \sim 6$ Å и найдём (примерно) максимальный линейный размер системы l. Для этого в окне VMD можно показать расстояние между терминальными атомами молекулы – азотом первой аминокислоты и карбоксильным углеродом последней. Выбрав показ расстояния между атомами из меню Mouse и кликнув последовательно на эти два атома можно получить текущее значение расстояния в ~ 27 Å. Таким образом, размер периодической ячейки будет $L = 27 + 2 \times 6 \sim 40$ Å.

В расширениях VMD есть утилита, позволяющая добавить ячейку растворителя (“Extensions” $\rightarrow$ “Modelling” $\rightarrow$ “Add Solvation Box”). В окне расширения необходимо указать пути к файлам pdb и psf, созданным на предыдущем шаге. Также следует указать размеры ячейки: в нашем случае минимальные и максимальные значения будут равны -20 и 20 Å по всем трём направлениям. Также стоит разрешить программе поворачивать молекулу для уменьшения объёма. После выполнения программы будут созданы обновлённые файлы координат и топологии (pdb и psf), которые будут содержать информацию как о полипептидной части системы (молекула полиаланина), так и о добавленной воде.

10.4.2 Ионы

В физиологических условиях в водном растворе присутствуют растворённые вещества, главным образом – ионы натрия и хлора. Хотя концентрация этих веществ может отли-

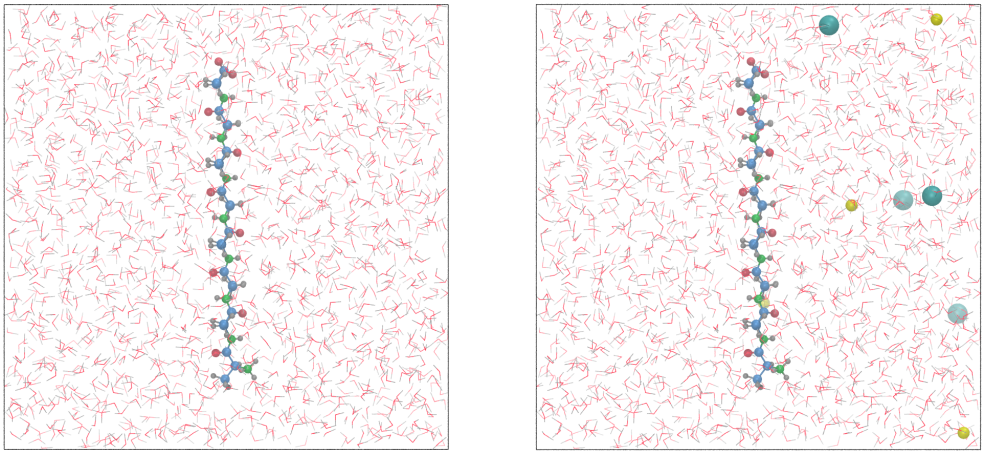


Рис. 36: Полиаланин, растворённый в воде без добавления ионов (слева) и с ними (справа). Вода показана линиями ковалентных связей, ионы натрия и хлора – жёлтыми и голубыми сферами соответственно. Чёрными линиями показана граница периодической ячейки.

чаться в различных частях организма, соли оказывают очень большое влияние на физиологические процессы. Отклонение концентрации от нормы может привести к серьёзным патологиям функции белков и, в некоторых случаях, даже к смерти организма. С точки зрения моделирования, отсутствие ионов может также привести к некорректной работе численных процедур. В частности, при использовании периодических условий необходимо, чтобы полный электростатический заряд моделируемой системы оставался нулевым. Для добавления ионов в нашу систему воспользуемся расширением “Add Ions” программы VMD. Так как наша система нейтральна, количество добавляемых отрицательно и положительно заряженных ионов должно быть одинаково, а в расширении “Add Ions” есть возможность указать желаемую концентрацию соли вместо явного указания желаемого количества ионов. После указания концентрации и выполнения программы, в системе некоторые молекулы воды будут заменены на атомы ионов (Рис. 36), в результате чего будут сгенерированы обновлённые файлы psf и pdb. Именно эти файлы описывают нашу систему (psf) и её начальную конформацию (pdb) и далее будут использованы программой NAMD в моделировании.

11 Начальные этапы моделирования

11.1 Минимизация энергии

Первым подготовительным этапом моделирования является минимизация энергии, которая позволяет избавиться от излишков энергии. Особенно этот этап актуален при использовании кристаллических структур, полученных в экспериментах с низким разрешением. Дело в том, что даже небольшие ошибки в определении позиции атомов могут привести к пересечению сфер Ван-дер-Ваальса и к большому значению внутренней энергии системы. При моделировании это может вызвать нефизические структурные изменения в молекуле, повышенное значение скоростей атомов и даже численную нестабильность расчёта. Для того, чтобы избавиться от излишка энергии, используется специальные протоколы моделирования. Самым простым таким протоколом является демпфирование или обнуление скоростей атомов после каждого шага интегрирования, а основная цель данного этапа – найти локальный минимум внутренней энергии.

Все подготовительные этапы требуют расчёта сил межатомных взаимодействий и будут выполняться с помощью программы NAMD, которую можно скачать с сайта Университета Иллинойса[5]. Интерфейс взаимодействия данной программы с пользователем представляет собой текстовый конфигурационный файл, в котором указаны все необходимые параметры моделирования. Большинство программ для молекулярного моделирования (например Gromacs, Amber, CHARMM) используют схожий интерфейс взаимодействия с пользователем. Таким образом, переход от одной программы к другой не представляет труда. Здесь мы остановимся на программе NAMD и перечислим необходимые для проведения минимизации энергии параметры конфигурационного файла.

Для того, чтобы сообщить программе информацию о моделируемой системе, необходимо указать путь к файлам топологии (psf) и начальных координат атомов (pdb), которые были созданы нами ранее при помощи программы VMD:

```
1 structure pala_ionized.psf
```

[5]http://www.ks.uiuc.edu/Research/namd/

```
2 coordinates pala_ionized.pdb
```

Для того, чтобы программа могла определить параметры силового поля, необходимо сообщить ей тип файла параметров и путь к нему. Важно, чтобы этот файл совпадал с использованным ранее при подготовке системы, особенно если в процессе подготовки были добавлены недостающие атомы.

```
1 paratypecharmm  on
2 parameters  par_all27_prot_na.prm
```

Далее рассмотрим возможные параметры конфигурации потенциала невалентных взаимодействий. Для некоторых типов атомов в файле параметров силового поля задаются особые параметры Ван-дер-Ваальса, если эти атомы связаны через 3 ковалентные связи (так называемые взаимодействия 1-4). Если два атома разделяет три ковалентные связи и особые параметры 1-4 для типа одного атома (обоих атомов) не заданы, то вместо не заданных параметров используются обычные параметры. Такой способ расчёта невалентных взаимодействий является наиболее популярным для большинства силовых полей и включается заданием значения "`scaled1-4`" обязательному конфигурационному параметру "`exclude`". Также можно, например, полностью исключить расчёт невалентных взаимодействий для атомов 1-4, задав значение "`1-4`" этому параметру. Электростатические взаимодействия для пар атомов 1-4 либо рассчитываются без изменений либо умножаются на заданную константу ("`1-4scaling`"):

```
1 exclude   scaled1-4
2 1-4scaling  1.0
```

Для ускорения расчёта, будем использовать обрезку потенциала невалентных взаимодействий на заданном расстоянии с функцией переключения (параметр "`switching`"), доводящей потенциал до нуля на заданной дистанции (параметр "`cutoff`" равный 12 Å). Функция переключения при этом будет включаться на расстоянии в 8 Å ("`switchdist`"):

```
1 switching on
2 switchdist  8.0
```

```
3 cutoff    12.0
```

Далее введём список соседей (список Верле), в который будут добавляться атомы, расстояние между которыми не превышает 13.5 Å. Частота обновления этого списка в шагах интегрирования задаётся при помощи параметра “`stepspercycle`”:

```
1 pairlistdist  13.5
2 stepspercycle 20
```

Электростатические взаимодействия в периодических граничных условиях считаются при помощи метода суммирования Эвальда. Так как этот метод использует дискретное преобразование Фурье, все заряды в системе распределяются по узлам дискретной сетки. На выбор количества узлов влияют два фактора. Для достижения лучшей производительности, количество узлов сетки должно быть кратно 2, 3 или 5 (это связано с особенностями реализации быстрого преобразования Фурье). Расстояние между узлами сетки сильно влияет на точность вычисления и, как правило, не должно сильно превышать 1 Å. Выберем для молекулы полиаланина, которая была растворена в водном кубе размерами 40×40×40, размеры сетки 48×48×48 (“`PMEGridSizeX`×`PMEGridSizeY`×`PMEGridSizeZ`”), что удовлетворяет обоим перечисленным условиям. Также следует задать точность преобразования Фурье (“`PMETolerance`”) и частоту точного перерасчёта потенциала электростатических взаимодействий (“`fullElectFrequency`”). Значение по умолчанию для последнего берётся равным частоте обновления списка соседей (параметр “`stepspercycle`”), хотя рекомендуется перерасчитывать электростатические взаимодействия не реже, чем через каждые 4 шага:

```
1 PME     on
2 PMETolerance    0.000001
3 PMEGridSizeX    48
4 PMEGridSizeY    48
5 PMEGridSizeZ    48
6 fullElectFrequency  4
```

Размеры периодической ячейки задаются как базовые векторы, на которые она натянута. В случае кубической ячейки эти векторы перпендикулярны и равны по длине линейному

размеру ячейки (40 Å для полиаланина). Расположение периодической ячейки задаётся при помощи параметра “cellOrigin” и в нашем случае совпадает с началом координат. Для того, чтобы программа переносила атомы системы в базовую ячейку, используется опция “wrapAll”:

```
cellBasisVector1 40.0  0.0  0.0
cellBasisVector2  0.0 40.0  0.0
cellBasisVector3  0.0  0.0 40.0
cellOrigin        0.0  0.0  0.0
wrapAll     on
```

Следует уделять особое внимание заданию параметров периодической ячейки, неверное использование которых является причиной большинства ошибок при моделировании. Единственным параметром из перечисленных выше, который будет изменяться на последующих этапах моделирования – координаты системы, которые определяют текущее состояние. Каждый следующий этап будет использовать последние координаты предыдущего. На данном этапе необходимо использовать алгоритм минимизации энергии, который включается при помощи параметра “minimization”. Процесс минимизации достаточно быстрый и для такой системы, как полиаланин, для достижения локального минимума потенциальной энергии достаточно порядка 10000 шагов:

```
minimization  on
numsteps  10000
```

В процессе симуляции на экран выводятся значения температуры системы и энергии потенциалов взаимодействия. Температура при этом рассчитывается через скорости атомов согласно распределению Максвелла-Больцмана. Для того чтобы задать частоту вывода в количестве шагов интегрирования, необходимо использовать параметр “outputenergies”:

```
outputenergies  100
```

Программа также может сохранять координаты всех атомов в бинарный файл формата dcd, который затем можно визуализировать в программе VMD. Частота сохранения координат и имя файла задаются при помощи следующих параметров:

```
dcdfreq    100
dcdfile    dcd/pala_min.dcd
```

Конечные координаты атомов потребуются на следующем этапе моделирования. Префикс имени файла, в который сохранятся координаты по окончании минимизации, задаётся параметром “`outputname`”. К префиксу будет добавлено расширение .coor для координат системы, NAMD также сохранит конечные скорости в файл с расширением .vel. Формат файла по умолчанию бинарный, что позволяет сохранять числа с плавающей точкой с большей точностью, но не гарантирует переносимость координат между разными компьютерами. Для использования знакомого нам текстового формата pdb можно использовать параметр “`binaryoutput`”:

```
binaryoutput  no
outputname  pdb/pala_min
```

Для того, чтобы начать минимизацию энергии, необходимо все перечисленные выше параметры собрать в одном конфигурационном файле и передать его в качестве параметра исполняемому файлу программы NAMD. В операционной системе Linux команда запуска выглядит следующим образом:

```
namd2 min.conf > min.out &
```

Здесь “`min.conf`” – имя конфигурационного файла, а стандартный вывод программы перенаправляется в файл “`min.out`”. Амперсанд (“&”) используется для того, чтобы освободить окно терминала для выполнения других команд. Например, можно следить за ходом работы программы при помощи команды “`tail`”:

```
tail -f min.out
```

Большинство современных компьютеров оснащено многоядерными процессорами, которые позволяют производить параллельный расчёт. Для этого потребуется многоядерная (multicore) версия программы NAMD, а запуск минимизации параллельно на двух ядрах осуществляется командой:

```
namd2_multicore +p2 min.conf > min.out &
```

Сохранение файла “`min.out`” удобно ещё и потому, что можно построить график зависимости энергии системы от шага интегрирования. Для этого нужно выделить из этого файла строчки, которые начинаются с “`ENERGY:`”, что в операционной системе Linux можно сделать при помощи следующей команды:

```
1 cat min.out | grep ENERGY: > min.dat
```

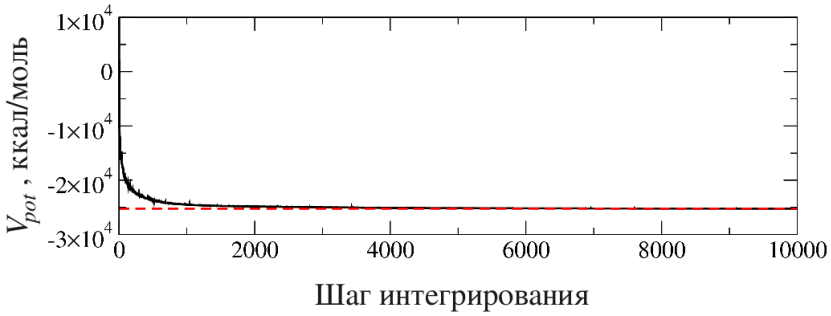


Рис. 37: График зависимости потенциальной энергии системы от шага интегрирования в процессе минимизации. Красной пунктирной линией показано конечное значение энергии.

Полученный файл (“`min.dat`”) содержит несколько столбцов со значениями температуры, энергий всех потенциалов, полной энергии, потенциальной и кинетической энергий, давления. Описание столбцов можно узнать, открыв файл стандартного вывода программы NAMD (“`min.out`”) и найдя строчку, которая начинается с “`ETITLE:`”. Потенциальная энергия сохраняется в 14-ом столбце, а шаг по времени – во втором, а график зависимости построенный по этим столбцам, показан на рисунке 37. Так как был использован протокол минимизации, программа NAMD сохраняла энергии на каждом шаге (вопреки указанному в конфигурационном файле значению в каждые 100 шагов).

Как видно из графика, потенциальная энергия резко падает, стремясь к ассимптотическому значению. Это значение – локальный минимум энергии. Чтобы загрузить траекторию в VMD, необходимо использовать файл psf, в котором содержится описание системы, и файл dcd, в который NAMD сохранила координаты. Можно заметить, что наиболее подвижны в процессе минимизации молекулы воды – именно не очень точное их начальное расположение является основным источником большого значения потенциальной энергии

в начале минимизации.

11.2 Нагрев системы

В физиологических условиях белки всегда находятся в движении. Как правило, их потенциальная энергия никогда не равна минимуму, а колеблется в его окрестностях из-за постоянного взаимодействия с нагретой до температуры среды водой. Полученная в результате минимизации энергии конформация, таким образом, должна быть "нагрета" до физиологической или экспериментальной температуры (обычно используется экспериментальная температура в 300 K). При нагреве системы температура увеличивается на заданное значение с заданной частотой, пока не будет достигнуто желаемое значение. Увеличение температуры обычно делается путём переназначения скоростей, распределение которых по атомам определяется статистикой Максвелла-Больцмана.

Конфигурационный файл для нагрева будет несколько отличаться от использованного в минимизации. Во-первых, в качестве начальной конформации необходимо задать конечные координаты этапа минимизации, то есть файл с префиксом, заданным параметром "`outputname`" на предыдущем шаге и с окончанием ".coor":

```
1 structure pala_ionized.psf
2 coordinates pdb/pala_min.coor
```

Если координаты на предыдущем этапе сохранялись в бинарном формате, то следует использовать "`bincoordinates`" вместо "`coordinates`". Так как топология системы не менялась, то файл psf остаётся тем же самым. Также не следует менять параметры силового поля, так как это приведёт к смещению минимума потенциальной энергии. Эти параметры, включая задание периодической ячейки, могут быть скопированы в конфигурационный файл нагрева системы.

Процесс нагрева системы отличается от обычного равновесного моделирования только тем, что скорости атомов постоянно переназначаются для достижения желаемой температуры. Уравнения движения интегрируются с использованием алгоритма Верле или алгоритма "прыжок лягушки". Таким образом, необходимо задать шаг по времени, для

которого обычно используют значение в 1 фс:

```
timestep    1.0
```

Скорость нагрева системы задаётся через начальную температуру (параметр “`temperature`”, в K), частоту переназначения скоростей (“`reassignFreq`”, в шагах интегрирования), значение, на которое температура будет каждый раз увеличена (“`reassignIncr`”, в K) и конечное значение температуры (“`reassignHold`”, в K), по достижению которого увеличение температуры прекратится:

```
temperature   0
reassignFreq    1
reassignIncr    0.01
reassignHold    300
```

Через перечисленные выше параметры определяется и количество шагов интегрирования, которое должно быть достаточно для достижения желаемой температуры. В нашем случае температура увеличивается на 0.01 K каждый шаг с 0 до 300 K, то есть количество шагов интегрирования должно быть не меньше 30000. Так как далее мы проведём этап эквилибрации системы, минимальная граница в 30000 является достаточным количеством шагов интегрирования на этапе нагрева системы:

```
numsteps    30000
```

Для переназначения скоростей атомов потребуется генератор (псевдо)случайных чисел, которому необходимо передать инициализирующее значение (зерно):

```
seed    21122012
```

Чтобы избежать перезаписи результатов минимизации, изменим также и имена файлов выходных данных:

```
outputenergies  100
dcdfreq   100
dcdfile   dcd/pala_heat.dcd
binaryoutput  no
```

```
5 outputname  pdb/pala_heat
```

После работы программы NAMD, запуск которой осуществляется аналогично этапу минимизации, можно построить график зависимости температуры и потенциальной энергии от времени (Рис. 38). Температура медленно растёт, достигая 300 K к концу симуляции. Как

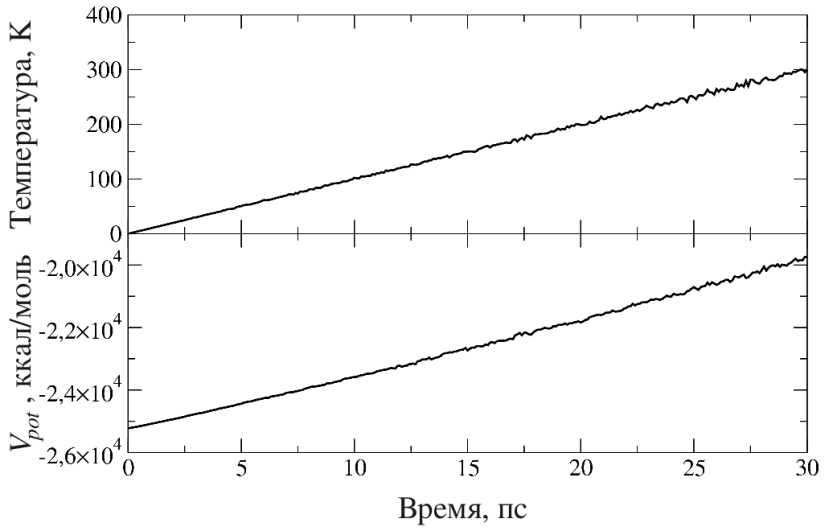


Рис. 38: График зависимости потенциальной энергии и расчётной температуры системы от времени в процессе нагрева.

видно из графика зависимости потенциальной энергии от времени, к концу симуляции её значение также увеличивается, то есть система выходит из состояния соответствующего локальному минимуму энергии.

11.3 Эквилибрация энергии

В процессе эквилибрации энергии скорости атомов также переназначаются, но в соответствии с постоянной температурой. Этот этап нужен для того, чтобы распределить энергию равномерно между кинетической и потенциальной составляющей. Кроме того, в процессе эквилибрации достигается случайная конформация, что помогает получить лучшую статистическую выборку термодинамических состояний. В принципе, эквилибрацию можно проводить в один этап с нагревом. Для этого достаточно увеличить количество

шагов, так как по достижению конечной температуры (параметр “reassignHold”) нагрев системы прекратится.

Параметры контроля температуры в процессе эквилибрации выглядят следующим образом:

```
temperature    300
reassignFreq     1
```

Также стоит увеличить количество шагов, изменить инициализирующее значение генератора случайных чисел и параметры выходных данных:

```
numsteps  100000
seed    31415926
outputenergies  100
dcdfreq   100
dcdfile   dcd/pala_equil.dcd
binaryoutput  no
outputname  pdb/pala_equil
```

В процессе эквилибрации скорости атомов периодически переназначаются согласно заданной температуре, что позволяет распределить энергию по потенциальной и кинетической компонентам. Как видно из графика на рисунке 39, значение потенциальной энергии выходит на плато. Последнее сигнализирует о том, что этап эквилибрации можно завершить. Несмотря на то, что скорости постоянно переназначаются, температура, определяемая из их распределения флуктуирует относительно равновесного значения в 300 K. Это происходит из-за того, что количество атомов в системе невелико (по сравнению с постоянной Авогадро). Как правило, увеличение размеров системы ведёт к уменьшению флуктуаций температуры на этапе эквилибрации, так как это даёт лучшее усреднение скоростей атомов.

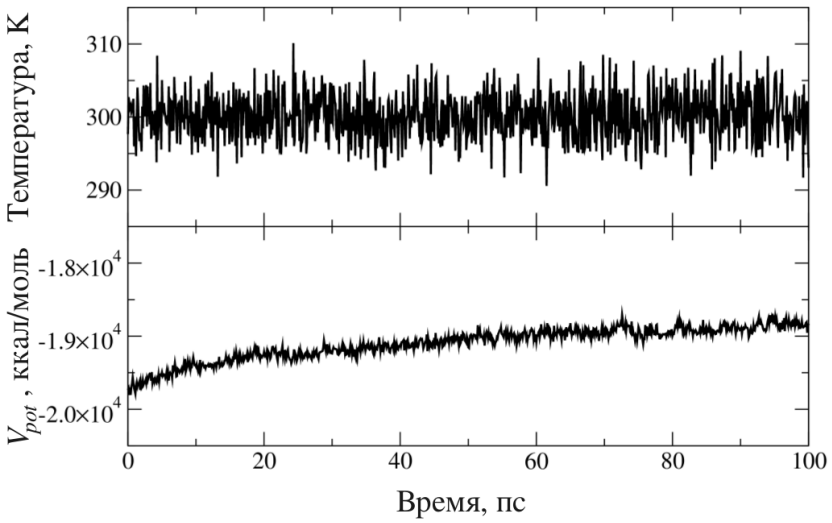


Рис. 39: График зависимости потенциальной энергии и расчётной температуры системы от времени для этапа эквилибрации энергии. Температура флуктуирует относительно заданного значения температуры, а потенциальная энергия несколько увеличивается, достигая в конце симуляции равновесного значения.

12 Моделирование системы и обработка результатов

12.1 Равновесное моделирование

После того, как все подготовительные этапы выполнены, можно переходить к моделированию системы. В данном примере рассматривается равновесное моделирование, конфигурационный файл для которого очень похож на использованный при эквилибрации энергии. Как и прежде, в качестве начальных координат будут использоваться конечные координаты предыдущего этапа, а файл топологии (psf) остаётся неизменным:

```
structure pala_ionized.psf
coordinates pdb/pala_equil.coor
```

Мы будем использовать стандартный шаг и экспериментальное значение температуры (1 фс и 300 K соответственно). Основным отличием равновесных симуляций от эквилибрации и нагрева является то, что скорости атомов переназначаться не будут. Хотя при длительном моделировании это может привести к смещению температуры от заданной изначально, в нашем случае длина траектории в 10^7 шагов (10 нс) позволяет просто отключить переназначение скоростей. Также изменим инициализирующее значение для генератора случайных чисел.

```
timestep    1.0
temperature   300
numsteps    10000000
seed      322223322
```

В параметрах выходных данных кроме имён файлов можно изменить и частоту сохранения энергии и координат:

```
outputenergies  1000
dcdfreq   1000
dcdfile   dcd/pala_quench.dcd
binaryoutput  no
outputname  pdb/pala_quench
```

12.2 Образование вторичной структуры

Основной целью данного упражнения было определить время, необходимое для формирования α-спирали в коротком пептиде. Существует несколько различных алгоритмов, которые позволяют охарактеризовать вторичную структуру белка из координат его атомов. Основные параметры, использующиеся в этих методах – значение торсионных углов пептидной (скелетной) связи и геометрия водородных связей. В программе VMD есть встроенная возможность определения вторичной структуры, которая использует алгоритм STRIDE [31].

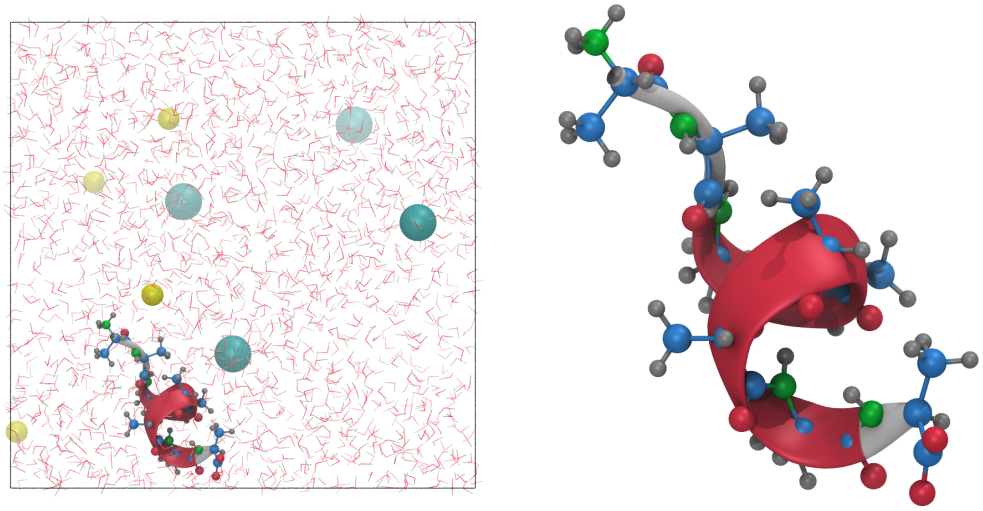


Рис. 40: α-спираль, образовавшаяся в процессе равновесного моделирования полиаланина.

Визуально наличие той или иной вторичной структуры можно проверить при помощи цветового кодирования, определяемого в графических репрезентациях молекулы в программе VMD. Стоит отметить, что в текущей версии программы, вторичная структура определяется лишь для одного набора координат системы, и для того, чтобы конфигурация вторичных структур переопределялась динамически, необходимо в TCL-консоли программы VMD выполнить скрипт “sscache” [6]. Хотя вторичная структура полиаланика постоянно изменяется, по истечению примерно 5 нс молекула большую часть времени находится в

[6]Скрипт доступен на сайте http://www.ks.uiuc.edu/Research/vmd/script_library/scripts/sscache/

α-конформации (Рис. 40).

Также, среди расширений VMD есть утилита “Timeline”, позволяющая показать динамику конфигурации вторичной структуры для всех аминокислот белка. В меню “Calculate” необходимо выбрать “Calc. Sec. Struc.” (англ. Calculate Secondary Structure – рассчитать вторичную структуру). По окончанию расчёта в окне утилиты появится диаграмма вторичных структур, цветовое кодирование которой приведено в нижнем левом углу окна (Рис. 41).

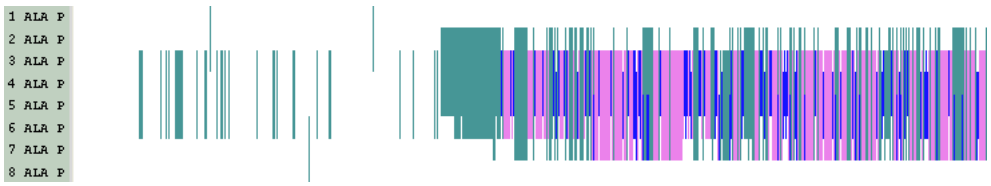


Рис. 41: Динамика вторичной структуры белка, рассчитанная при помощи утилиты “Timeline” программного пакета VMD. По горизонтальной оси отложено время моделирования, по вертикальной – восемь аминокислот молекулы. Фиолетовым показаны α-спирали.

Часть IV

Графические процессоры (ГП)

13 Вычисления общего характера при помощи графических процессоров

Благодаря растущим потребностям рынка компьютерных игр и обработки изображений программируемые графические процессоры эволюционировали в высокопараллельные многопоточные вычислительные устройства, обладающие огромными вычислительными возможностями и большой пропускной способностью памяти. Основным отличием архитектуры графических процессоров (ГП) от центральных процессоров (ЦП) является то, что графические процессоры изначально проектировались для высокопараллельных вычислений при обработке трёхмерных изображений. ГП изначально нацелены на более быструю обработку больших объёмов данных, а не на их кеширование и более сложный поток команд. Схематично это показано на рисунке 42.

Большое количество логических элементов кеш-памяти и контроллера на ЦП позволяет им быстро выполнять сложные вычислительные процедуры, имея быстрый доступ к данным в памяти. На ГП большее количество логических элементов отведено для вычислений, а модули контроля процедур и кеша уменьшены (Рис. 42). Благодаря этому, плотность Арифметических Логических Устройств (АЛУ) на ГП существенно выше, чем на ГП. Для того, чтобы загрузить все АЛУ вычислениями, ГП должен выполнять много операций одновременно. Это осуществляется при помощи вычислительных потоков, каждый из которых выполняет те же самые арифметические операции, но на разных элементах массива данных. Например, на ЦП сложение двух векторов размерности M необходимо осуществлять в цикле. При этом все элементы складываются последовательно один за другим. На ГП, подобная процедура может быть выполнена в M независимых потоках, каждый из который получает один элемент результирующего вектора. Таким образом, все элемен-

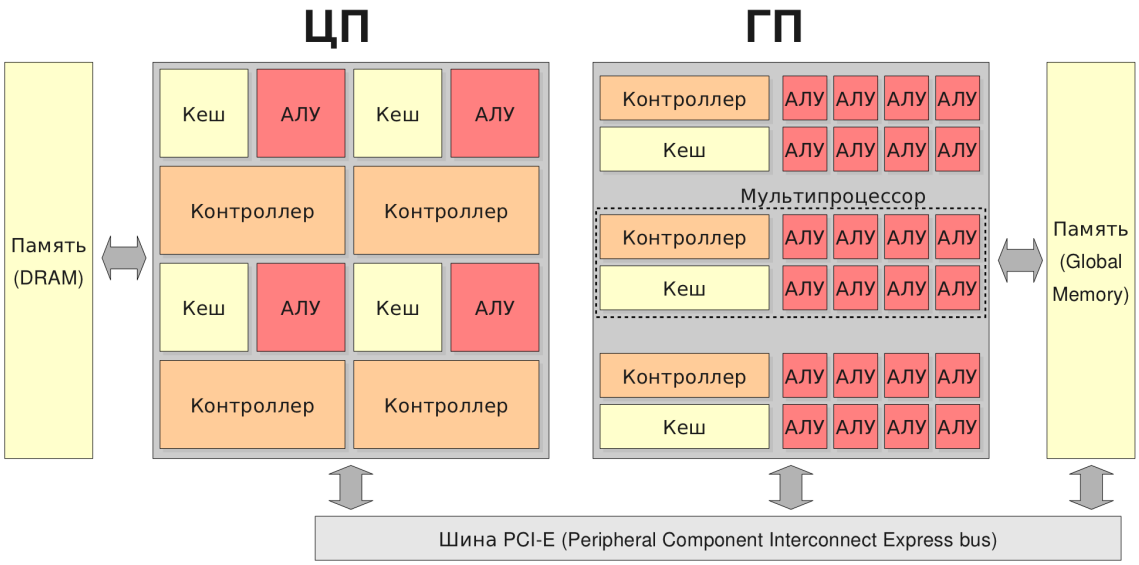


Рис. 42: Архитектура центрального (слева) и графического (справа) процессоров. На ЦП, значительная часть поверхности микро-чипа занята кеш-пямятью и контроллером. ЦП также оснащён изменяемым количеством памяти DRAM (Dynamic Random Access Memory). Арифметические Логические Устройства (АЛУ) на ГП, сгруппированные в мультипроцессоры, каждый из который обладает своими контроллером и кеш-памятью. Это позволяет существенно увеличить плотность вычислительных единиц на микро-чипе. ГП обладает отдельной памятью DRAM, которую обычно называют глобальной памятью устройства. Взаимодействие между ЦП и ГП осуществляется через шину PCI Express.

ты суммы могут быть получены одновременно и время подобной операции соответствует времени выполнения одного сложения против M сложений на ЦП.

ГП особенно хорошо подходят для решения вычислительных задач, которые можно разложить на параллельные вычисления на массиве данных – одни и те же операции производятся на большом количестве элементов в памяти. При этом, чем больше отношение количества арифметических операций к количеству операций чтения из памяти и записи в память, тем более эффективна будет реализация алгоритма на ГП. Так как производимые над разными элементами массива данных операции идентичны, наличия сложного контроллера не требуется. При большом количестве выполняемых операций, задержка в доступе к данным может быть скрыта выполнением операций над уже доступными элементами массива, что позволяет существенно сократить размер кеш-памяти.

14 Платформа CUDA

Технология CUDA, представленная компанией NVIDIA в ноябре 2006 года, является программной платформой для параллельных вычислений общего характера, предоставляя возможность использовать возможности ГП в любых высокопараллельных вычислительных задачах, решая их гораздо эффективнее, чем это возможно с применением ЦП. Кроме того, CUDA является расширением языка C и поставляется с полным инструментарием, необходимым разработчику программного обеспечения.

Современные ЦП и ГП многоядерны, то есть повсеместно используемые вычислительные устройства параллельны. Более того, в то время как производители ЦП уже давно столкнулись с технологическими ограничениями, препятствующими росту производительности одного вычислительного ядра, производительность параллельных устройств пропорциональна количеству вычислительных ядер и подчиняется закону Мура. Основной сложностью является разработка программного обеспечения, которое могло бы получать пропорциональный прирост производительности при выполнении на новых высокопараллельных процессорах с постоянно растущим числом вычислительных ядер. Данная задача уже была успешно решена в приложениях, работающих с трёхмерной графикой.

Программная платформа CUDA построена вокрух иерархии вычислительных потоков, которые разделяются на группы. На аппаратном уровне это соответствует разделению ГП на мультипроцессоры – больше групп потоков, больше мультипроцессоров может быть задействовано при выполнении программы. При наличии достаточного параллелизма в задаче, скорость её выполнения будет пропорциональна вычислительным возможностям ГП. Кроме того, платформа CUDA достаточна проста в изучении для людей, знакомых со стандартными языками программирования, такими как C и CUDA.

15 Программная модель

15.1 Вычислительные ядра

Диалект CUDA для языка C позволяет программисту определять специальные функции, называемые ядрами, которые при вызове выполняются параллельно N раз в N вычислительных потоках. Ядро определяется при помощи модификатора `__global__`, а количество желаемых потоков задаётся при помощи синтаксиса `<<<...>>>` – конфигурации выполнения ядра. Каждый поток, выполняющий ядро, обладает уникальным идентификатором потока, который доступен внутри ядра через встроенную переменную `threadIdx`. Например, следующая программа выполняет сложение двух векторов a и b размера N и сохраняет результат в векторе c:

```
1  __global__ void vecAdd(float* a, float* b, float* c){
2    int i = threadIdx.x;
3    c[i] = a[i] + b[i];
4  }
5
6  int main(){
7    ...
8    vecAdd<<<1, N>>>(a, b, c);
9    ...
10 }
```

15.2 Иерархия потоков

Индекс потока `threadIdx` – трёхмерный вектор, а потоки можно идентифицировать, используя трёх-, двух- или одномерный индекс, формируя трёх-, двух- или одномерные блоки потоков. Это позволяет распределять вычисления по элементам объёма, матрицы или вектора, выбирая соответствующую размерность индекса потока. Так как все потоки блока выполняются на одном мультипроцессоре, доступные им ресурсы (кеш-память,

регистры) ограничены. Поэтому, количество потоков в блоке ограничено и зависит от поколения ГП (на графических картах с архитектурой Fermi это ограничение – 1024 потока на блок).

Обычно ядро выполняется в множестве одинаковых блоков потоков. Общее количество потоков в таком случае определяется как произведение количества блоков на количество потоков в каждом из них. Как и потоки внутри блоков, блоки объединяются в одно-, двух- или трёхмерную сетку блоков. Блоки в ядре можно идентифицировать с использованием переменной `blockIdx`, а их размер – через переменную `blockDim`. Таким образом, общий индекс потока на сетке в одномерном случае можно определить как `blockIdx*blockDim+threadIdx`. Количество потоков внутри блока и количество блоков может задаваться переменными типа `int` или `dim3` внутри скобок `<<<...>>>` при запуске ядра. Предыдущий пример с разбиением потоков на блоки будет записываться как:

```
1  __global__ void vecAdd(float* a, float* b, float* c){
2    int i = blockIdx*blockDim+threadIdx;
3    c[i] = a[i] + b[i];
4  }
5
6  int main(){
7    ...
8    int threadsPerBlock = 128;
9    int numberOfBlocks = N/128;
10   vecAdd<<<numberOfBlocks, threadsPerBlock>>>(a, b, c);
11   ...
12 }
```

Блоки потоков должны выполняться независимо, а порядок их выполнения заранее неизвестен. Более того, в CUDA нет возможности синхронизировать выполнение потоков из разных блоков. Внутри блока потоки могут передавать данные через разделяемую память, а для их барьерной синхронизации существует встроенная функция `__syncthreads()`. Если при выполнении команд поток встречает вызов функции `__syncthreads()`, его выполне-

ние приостанавливается до тех пор, пока все остальные потоки внутри того же блока не достигнут вызова этой функции.

15.3 Иерархия памяти

Очень важной частью любой программы, использующей CUDA, является управление памятью. Часть такой программы выполняется на ЦП, а часть – на ГП, и эти устройства обладают своей областью памяти. Для выделения памяти на ЦП можно использовать стандартные процедуры языка C, такие как `calloc(...)`. Для выделения памяти на устройстве используется процедура `cudaMalloc(...)` из диалекта CUDA для C. Для копирования данных между памятью ЦП и ГП используется процедура `cudaMemcpy(...)`. Вернёмся к примеру со сложением векторов. Предположим, что векторы заполняются на ЦП, затем данные копируются на ГП, векторы складываются, результат возвращается в память ЦП и выводится на экран:

```
1  int main(){
2    // Allocate memory on CPU (host)
3    float* h_a = (float*)calloc(N, sizeof(float));
4    float* h_b = (float*)calloc(N, sizeof(float));
5    float* h_c = (float*)calloc(N, sizeof(float));
6    ...
7    // Fill arrays h_a and h_b with data
8    ...
9    // Allocate memory on GPU (device)
10   float* d_a; float* d_b; float* d_c;
11   cudaMalloc((void**)&d_a, N*sizeof(float));
12   cudaMalloc((void**)&d_b, N*sizeof(float));
13   cudaMalloc((void**)&d_c, N*sizeof(float));
14   // Copy data from CPU to GPU memory
15   cudaMemcpy(d_a, h_a, N*sizeof(float), cudaMemcpyHostToDevice);
16   cudaMemcpy(d_b, h_b, N*sizeof(float), cudaMemcpyHostToDevice);
```

```
17    // GPU kernel launch
18    int threadsPerBlock = 128;
19    int numberOfBlocks = N/128;
20    vecAdd<<<numberOfBlocks, threadsPerBlock>>>(d_a, d_b, d_c);
21    // Copy the result back to CPU memory
22    cudaMemcpy(h_c, d_c, N*sizeof(float), cudaMemcpyDeviceToHost);
23    // Print the result on the screen
24    for(i = 0; i < N; i++){
25      printf("%f\n", h_c[i]);
26    }
27    // Deallocate the memory
28    free(h_a); free(h_b); free(h_c);
29    cudaFree(d_a); cudaFree(d_b); cudaFree(d_c);
30  }
```

В приведённых процедурах указатели на данные, находящиеся в памяти ЦП, обозначены префиксом `h_` (от англ. host), а на данные в памяти ГП – префиксом `d_` (от англ. device). Важно понимать, что при прямом обращении к данным по указателю на память ГП из участка программы, выполняющейся на ЦП, произойдёт ошибка. Тут можно привести аналогию с почтовыми адресами: для находящихся в одном городе адрес в другом городе не имеет смысла.

Память ГП доступна из кода ЦП при помощи процедур диалекта CUDA. Таким образом, процесс выполнения программы управляется ЦП, а ГП отведена роль со-процессора. Важно отметить, что копирование данных из памяти ЦП в память ГП осуществляется через интерфейс PCI-Express, который обладает низкой пропускной способностью по сравнению с шиной доступа к памяти. Большое количество операций копирования данных может существенно замедлить выполнение программы, поэтому для получения высокой производительности необходимо минимизировать перенос данных.

При написании программ, работающих на ЦП, порядок и способ обращения к памяти не играет ключевой роли в производительности программы. На ГП порядок и способ обра-

щения к памяти является одним из главных факторов, влияющих на производительность. Происходит это из-за малого объёма кеш-памяти, а также из-за того, что множество потоков может обращаться к памяти одновременно. Поэтому, в диалекте CUDA программист имеет больше возможностей контролировать потоки данных из одной области памяти ГП в другую.

На ГП есть несколько различных областей памяти (Таблица 2). Во-первых каждый мультипроцессор оснащён набором регистров и кешем. Регистры распределяются между потоками, а кеш память может быть использована либо для кеширования элементов глобальной памяти либо как разделённая память, доступная всем выполняющимся на этом мультипроцессоре потокам. Эти области памяти располагаются на чипе ГП. В памяти, расположенной на плате графической карты, располагается глобальная память, доступ к которой может осуществляться как из любого потока выполняющегося на ГП, так и с ЦП при помощи специальных команд CUDA (напр. `cudaMemcpy(...)`. На современных ГП доступы к глобальной памяти кешируются. Кроме того, существует область постоянной памяти, которая всегда кешируется. Чтение из глобальной памяти также можно осуществлять при помощи текстурных ссылок, которые кешируются пространственно (вместе с читаемым элементом в кеш заносятся и элементы, его окружающие в памяти).

Таблица 2: Типы памяти в CUDA.

Тип	Тип (англ.)	Расположение	Кеширование	Доступ	Доступна	Время жизни
Регистр	Register	На чипе		чтение/запись	из потока	Поток
Разделённая	Shared	На чипе		чтение/запись	из блока	Блок
Глобальная	Global	На плате	Да/Нет	чтение/запись	глобально	Приложение
Постоянная	Constant	На плате	Да	только чтение	глобально	Приложение
Текстурная	Texture	На плате	Да	только чтение	глобально	Приложение

Иерархическая структура данных на ГП открывает широкие возможности по оптимизации программного кода. Во-первых, количество регистров на мультипроцессоре ограничено, поэтому, чтобы максимизировать количество одновременно выполняющихся на мультипроцессоре потоков, программист должен контролировать количество используемых локальных величин внутри ядра. Во-вторых, разделённая память может быть использована

для явного кеширования данных. В-третьих, количество обращений к глобальной памяти должно быть минимизировано, либо распределено по программному коду так, чтобы все потоки могли выполнять арифметические операции в процессе ожидания данных. Также, в силу наличия аппаратного кеша, обращение к глобальной памяти должно быть локализовано – соседние потоки должны обращаться к соседним ячейкам памяти. Доступ к глобальной памяти устройства должен осуществляться сопряжённо: следующие друг за другом потоки должны использовать следующие друг за другом элементы массива из глобальной памяти устройства. Это позволяет устройству склеивать несколько доступов к памяти в один. Выравнивание потоков относительно массивов с данными имеет много тонкостей и зависит от архитектуры (поколения) ГП. Для более детального ознакомления со способами доступа к памяти и её выравнивания читателю следует изучить руководство программиста, поставляемое с CUDA. Существенным фактором, который может ограничить производительность программы, является скорость передачи данных от ЦП к ГП, которая ограничена шиной PCI-Express. Программист должен, по возможности, избегать копирования данных между ЦП и ГП. Этого можно добиться переносом всего программного кода на ГП, используя ЦП лишь для управления потоком вычислений и вывода результатов.

16 Диалект CUDA для C

16.1 Модификаторы функций

Модификаторы функций сообщают компилятору, может ли данная функция выполняться на ГП или на ЦП и можно ли её вызвать с ЦП или с ГП. Модификатор `__device__` говорит о том, что функция может выполняться только на устройстве и вызывается только с устройства. Модификатор `__global__` ставится перед функцией, которая является вычислительным ядром (вызывается с ЦП и выполняется на ГП). При вызове данной функции необходимо указывать конфигурацию её запуска (количество блоков потоков и количество потоков в каждом из них при помощи синтаксиса `<<<...>>>`). Функции с данным модификатором должны возвращать `void`. Модификатор `__host__` говорит о том, что данная функция может вызываться только с ЦП, где она и выполняется. Отсутствие всех трёх модификаторов эквивалентно присутствию модификатора `__host__`. Комбинацию модификаторов `__host__` и `__device__` можно использовать одновременно. В этом случае код будет компилироваться дважды – для ЦП и для ГП.

16.2 Встроенные векторные типы

В CUDA используются встроенные векторные типы для всех типов целочисленных переменных и переменных с плавающей точкой. Все эти типы представляют собой структуры, первая, вторая, третья и четвёртая компоненты которых доступны через поля `x`, `y`, `z` и `w` соответственно. Например, переменная типа `float3` будет иметь три поля – `x`, `y` и `z`. Для всех векторных типов определён конструктор, имя которого соответствует типу: `make_<name>`. Например, для того, чтобы создать вектор с компонентами (x, y, z) можно вызвать конструктор:

```
1 float x, y, z;
2 float3 r = make_float3(x, y, z);
```

16.3 Математические функции

В процедурах, выполняемых на ГП, кроме стандартных математических функций (например `sinf(alpha)`) доступны их более быстрые, но менее точные встроенные аналоги. Последние обозначаются двойным подчёркиванием перед названием функции (например `__sinf(alpha)`. При использовании данных функций нужно удостовериться, что предоставляемая ими точность устраивает при выполнении тех или иных операций. В некоторых особых случаях результаты, получаемые с использованием встроенных функций, могут существенно отличаться от желаемых, поэтому правильным способом является замена функций на встроенные в тех местах, где это не повлияет на результат. Для того, чтобы компилятор использовал встроенные функции вместо стандартных, можно при компиляции программы указать опцию `-use_fast_math`.

17 Устройство ГП

17.1 Мультипроцессоры

Ядро ГП компании NVIDIA представляет собой набор многопоточных мультипроцессоров. Когда программа, написанная на CUDA, вызывает вычислительное ядро, блоки потоков нумеруются и распределяются по мультипроцессорам устройства. Потоки из одного блока выполняются параллельно на одном мультипроцессоре, которому могут быть назначены сразу несколько блоков потоков. Как только один из блоков заканчивает выполнение вычислительных процедур, мультипроцессор начинает выполнение следующего блока потоков.

Мультипроцессоры могут выполнять сотни потоков одновременно. Для того, чтобы управлять таким большим количеством потоков, мультипроцессоры используют уникальную архитектуру: одиночный поток команд, множественные потоки выполнения (ОКМП; от англ. Single Instruction, Multiple Thread, SIMT). Инструкции выполняются параллельно с использованием собственного параллелизма инструкций, либо последовательно с использованием параллелизма множества потоков.

17.2 Архитектура ОКМП

Мультипроцессор создаёт группы по 32 потока в каждой (англ. warp). Все потоки одной такой группы начинают выполняться одновременно с использованием общего набора команд. Так как у каждой группы свой набор регистров, группы выполняются независимо друг от друга. Если мультипроцессор выполняет более одного блока потоков, то все блоки делятся на группы по 32 потока и далее выполняются в этих группах. Так как группа потоков выполняет один и тот же набор команд, для достижения максимальной производительности следует избегать ветвлений внутри группы. В случае возникновения последнего, некоторые потоки будут вынуждены простаивать, пока другие выполняют процедуры, предусмотренные их ответвлением.

Основным отличием архитектуры ОКМП от архитектуры ОКМД (одиночный поток

команд, множественный поток данных; от англ. Single Instruction, Multiple Data, SIMD) является то, что программисту даётся возможность контролировать не только общие для всех данных команды, но и выполнение каждого потока по-отдельности. Таким образом программисту открываются большие возможности по оптимизации кода: существенного прироста производительности можно достичь, если распределять потоки по группам из 32 штук с учётом возникновения в них ветвлений.

Часть V

Пример: моделирование кислорода в равновесии

18 Постановка задачи

Для того, чтобы разобрать в подробностях способы реализации численных методов молекулярной динамики на графических процессорах, рассмотрим задачу моделирования кислорода в равновесном состоянии. При этом ковалентная связь между атомами кислорода внутри молекулы O_2 будет описываться при помощи гармонического потенциала, а атомы из разных молекул будут взаимодействовать через потенциал Леннарда-Джонса (взаимодействия Ван-дер-Ваальса). Так как атомы кислорода в молекуле O_2 не заряжены (частичный заряд равен нулю), то электростатические взаимодействия в данной системе отсутствуют. Для сохранения объёма системы мы будем использовать периодические граничные условия, применение которых скажется только на формулах расчёта расстояния между атомами: в силу отсутствия дальнодействующего электростатического потенциала, возможен прямой расчёт невалентных взаимодействий с применением радиуса обрезки. Таким образом, потенциальная энергия системы будет описываться при помощи следующей функции координат атомов:

$$V = V_{bond} + V_{LJ} = \sum_{bonds} \frac{1}{2} K_s (r_{ij} - b_0)^2 + \sum_{i,j} \varepsilon \left[\left(\frac{\sigma}{r_{ij}} \right)^{12} - 2 \left(\frac{\sigma}{r_{ij}} \right)^{6} \right], \tag{24}$$

где $r_{ij} = |r_j - r_i|$, а значения параметров перечислены в таблице 4.

Важным моментом является выбор системы единиц. Очевидно, что более удобными единицами измерения являются величины, так или иначе характеризующие молекулярную систему (вряд ли кому-то придёт в голову измерять расстояния между атомами в

метрах). В данном примере мы будем использовать единицы измерения МД, идентичные используемым в программе Gromacs [18]:

Таблица 3: Система единиц МД.

Величина	Единицы измерения
Длина	нм=10^{-9}м
Масса	г/моль=1.6605402×10^{27}кг
Время	пс=10^{-12}с
Температура	K

Отметим, что в данной системе единиц энергия измеряется в (г/моль)нм^2/пс^2 = 10^3 (кг/моль)м^2/с^2 = кДж/моль, а сила – в (кДж/моль)/нм. Постоянная Больцмана k_B = 8.314462×10^{-3}кДж/(моль·К). Параметры взаимодействия между атомами кислорода в данной системе единиц приведены в Таблице 4.

Таблица 4: Параметры взаимодействий для атомов молекулы кислорода O_2.

Параметр	Значение	Описание
m	15.999 г/моль	Молярная масса атома кислорода
b_0	0.121 нм	Равновесное расстояние между атомами в молекуле O_2
K_s	518816 кДж/моль·нм^2	Постоянная упругости для ковалентной связи O=O
σ	0.302905564168 нм	Расположение минимума энергии потенциала Леннарда-Джонса
ε	0.50208 кДж/моль	Глубина минимума потенциала Леннарда-Джонса

Далее мы подробно и по порядку разберём все вычислительные процедуры для моделирования данной системы. В ходе реализации мы будет использовать стандартные простые типы CUDA (`float2`, `float3` ...), а также некоторые предопределённые операции с этими типами (например, покомпонентная разность двух векторов будет определена через оператор “−” (минус) для двух переменных типа `float3`).

19 Начальная конформация системы

19.1 Случайное расположение атомов

Рассмотрим кубическую периодическую систему, в которой границы задаются как $0 < \{x, y, z\} < L$. Пусть количество атомов равно $N = 2k$, где k-целое (k - число молекул O_2), и пусть каждый атом с чётным индексом i ($i = 0, 2, 4 \dots N-2$) связан валентно со следующим атомом с индексом j ($j = 1, 3, 5 \dots N-1$). Для размещения следующего чётного атома i нужно задать случайные координаты $0 < \{x_i, y_i, z_i\} < L$ таким образом, что сферы Ван-дер-Ваальса этого атома не пересекаются со сферами Ван-дер-Ваальса атомов, уже добавленных в систему (электронные облака атомов не пересекаются). При этом следует учитывать, что в системе с периодическими граничными условиями расстояние между атомами i и j задаётся как минимальное расстояние от атома i до атома j и его периодических отображений, то есть:

$$\vec{r}_{ij} = \vec{r}_j - \vec{r}_i - [(\vec{r}_j - \vec{r}_i)/L] \cdot L, \tag{25}$$

где квадратные скобки обозначают округление. Программная реализация данного алгоритма:

```
1 inline __host__ __device__ float3 getVector(float3 ri, float3 rj, float L){
2   float3 dr = rj -ri;
3   dr -= rint(dr/L)*L;
4   return dr;
5 }
```

Кроме модификатора встраивания процедуры (`inline`) здесь использованы модификаторы из диалекта CUDA – `__host__` и `__device__`, благодаря которым процедура будет скомпилирована для выполнения как на ЦП так и на ГП и её `getVector(...)` можно будет использовать в обоих частях кода. Расстояние между двумя атомами в условиях периодических граничных условий определяется длиной этого вектора, то есть:

```
1 inline __host__ __device__ float getDistance(float3 ri, float3 rj, float L){
```

```
  float3 dr = getVector(ri, rj, L);
  return len(dr);
}
```

Где `len(float3 r) = sqrtf(r.x*r.x + r.y*r.y + r.z*r.z);`. Кроме того, нам потребуется процедура перемещения атомов в область $0 < \{x, y, z\} < L$ через периодические граничные условия, которую можно задать как:

```
float3 transferPBC(float3 r){
  if(r.x > L){
    r.x -= L;
  } else
  if(r.x < 0){
    r.x += L;
  }
  ...
  return r;
}
```

Расположение валентно связанного с атомом i атома j при этом можно задать случайным углом ориентации связи O=O, расположив атом j на равновесном расстоянии от атома i. В качестве генератора случайных чисел мы будем использовать ran2 ([32], Приложение 1).

```
void placeAtoms(){
  float3 r1, r2, dr;
  int i, j;
  for(i = 0; i < N/2; i++){
    float mindist = 0.0;
    while(mindist < 2.0*sigma){
      r1.x = L*ran2(&rseed); r1.y = L*ran2(&rseed); r1.z = L*ran2(&rseed);
      dr.x = ran2(&rseed); dr.y = ran2(&rseed); dr.z = ran2(&rseed);
      float norm = len(dr);
      dr /= norm;
```

```
11        dr *= b0;
12        r2 = r1 + dr;
13        r2 = transferPBC(r2, L);
14        mindist = FLT_MAX;
15        for(j = 0; j < i; j++){
16          float dist = getDistance(r1, r[2*j], L);
17          if(dist < mindist){
18            mindist = dist;
19          }
20        }
21      }
22      r[2*i] = r1;
23      r[2*i+1] = r2;
24    }
25  }
```

В данной вычислительной процедуре первому атому из молекулы кислорода присваиваются случайные координаты (`r1`), а второму (`r2`) – координаты, удалённые от первого на равновесную длину ковалентной связи O=O. При этом, для каждой пары атомов случайное число бросается до тех пор, пока минимальное расстояние между одним из атомов пары и атомами, уже добавленными в систему, не будет превышать двух равновесных расстояний потенциала Леннарда-Джонса (`2.0*sigma`). Начальное расположение атомов показано на рисунке 43.

19.2 Распределение скоростей

Начальные значения скоростей определяются температурой системы. Для того, чтобы их задать, мы будем использовать случайные числа, распределённые по Гауссу. Обычно, генераторы случайных чисел возвращают равномерно распределённые случайные числа, которые затем можно преобразовать в случайные числа, распределённые нормально при

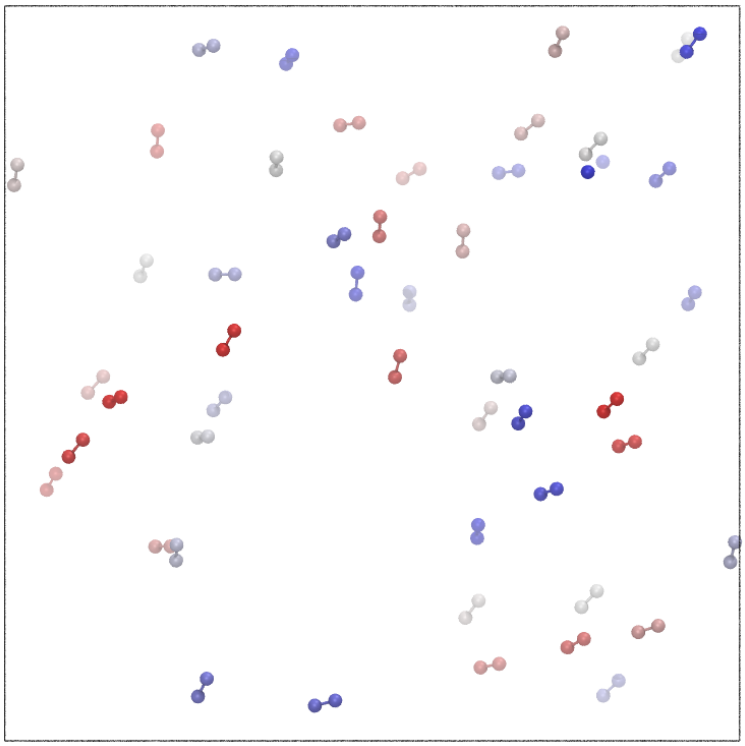


Рис. 43: Случайное расположение 50 молекул кислорода в кубе $4 \times 4 \times 4$ нм. Чёрными линиями показаны граничные условия. Атомы показаны в представлении CPK, цвет соответствует порядковому номеру атома.

помощи преобразования Бокса-Мюллера [32], которое задаётся следующим образом. Пусть u_1 и u_2 – независимые случайные величины, равномерно распределённые на отрезке [-1, 1]. Вычислим $s = u_1^2 + u_2^2$. Если окажется, что $s > 1$ или $s = 0$, то значения u_1 и u_2 следует сгенерировать заново. Как только выполнится условие $0 < s \leq 1$, можно рассчитать две независимые случайные величины n_1 и n_2, удовлетворяющие нормальному распределению с математическим ожиданием 0 и дисперсией 1. Для этого необходимо использовать следующие уравнения:

$$\begin{aligned} n_1 &= u_1 \sqrt{\frac{-2 \ln s}{s}} \\ n_2 &= u_2 \sqrt{\frac{-2 \ln s}{s}} \end{aligned} \tag{26}$$

Так как обычно случайные величины требуются одна за другой, а преобразование Бокса-Мюллера создаёт сразу две нормально распределённые величины (из двух равномерно распределённых), имеет смысл сохранять одну из полученных нормально распределённых величин чтобы возвратить её при повторном обращении:

```
inline double gasdev(int *idum) {
  static int iset = 0;
  static double gset;
  double fac, rsq, v1, v2;
  if(iset == 0){
    do {
      v1 = 2.0*ran2(idum) - 1.0;
      v2 = 2.0*ran2(idum) - 1.0;
      rsq = v1*v1+v2*v2;
    } while(rsq >= 1.0 || rsq == 0.0);
    fac = sqrt(-2.0*log(rsq)/rsq);
    gset = v1*fac;
    iset = 1;
    return v2*fac;
  } else {
    iset = 0;
    return gset;
  }
}
```

Согласно распределению Максвелла, компоненты скоростей частиц распределены нормально, с математическим ожиданием 0 и дисперсией $\sqrt{k_B T/m}$. Таким образом, чтобы получить компоненту скорости частицы массой m при температуре T, необходимо получить случайную величину n (Уравнение 26) и умножить её на $\sqrt{k_B T/m}$. В программе это можно сделать при помощи следующей простой процедуры:

```
void assignVelocities(){
```

```
2   for(i = 0; i < N; i++){
3     double var = sqrt(kB*T/m);
4     v[i].x = var*gasdev(&seed);
5     v[i].y = var*gasdev(&seed);
6     v[i].z = var*gasdev(&seed);
7   }
8 }
```

19.3 Сохранение состояния системы

Рассматриваемая нами система может быть описана $6 * N$ обобщёнными координатами – тремя компонентами координат ($\{x, y, z\}$) и тремя компонентами скоростей ($\{v_x, v_y, v_z\}$) для каждой из N частиц. Значения этих величин необходимо сохранять для того, чтобы в дальнейшем можно было проанализировать поведение системы. Кроме того, формат данных должен быть общепринятым, чтобы его можно было использовать в стандартных программах визуализации и анализа, таких как VMD. В случае сохранения траекторий, когда в файле содержится много состояний системы, огромную роль играет также и размер этого файла. Поэтому, обычно для этих целей используются бинарные форматы (например – .dcd в NAMD и CHARMM или .trr в Gromacs). В рамках нашего примера мы остановимся на текстовом формате .xyz. Конечно, можно было бы использовать и описанный ранее формат .pdb, но он содержит много информации, которая в нашем случае не обязательна. Формат .xyz гораздо проще: он содержит одну строку с целым числом, равным количеству атомов в молекулярной системе (N), одну строку комментариев и N строк с разделёнными пробелами или табуляцией именами атомов и их координатами (в Ангстремах). Например файл формата .xyz для координат атомов одной молекулы кислорода выглядит следующим образом:

```
2
Oxygen molecule (in Angstroms)
O       -0.605000        0.000000        0.000000
```

```
O       0.605000        0.000000        0.000000
```

При сохранении нескольких состояний все строки файла повторяются. Для системы, состоящей только из кислорода, имена всех атомов одинаковы. Координаты атомов сохраняются в Ангстремах, поэтому, в нашем случае их значения необходимо преобразовать из нм, умножив значения на 10:

```
1  void writeXYZ(const char* filename, float3* data, int N){
2    FILE* file = fopen(filename, "w");
3    fprintf(file, "%d\n", N);
4    fprintf(file, "Oxygen\n");
5    int i;
6    for(i = 0; i < N; i++){
7      fprintf(file, "O\t%f\t%f\t%f\n",
8        10.0*data[i].x, 10.0*data[i].y, 10.0*data[i].z);
9    }
10   fclose(file);
11 }
```

Данная процедура построена таким образом, что она может сохранять не только координаты атомов, но и их скорости. Кроме того, простое изменение индикатора открытия файла на “a” (append – добавить) позволяет сохранять файлы, содержащие более одного набора координат. Далее мы используем эту процедуру в отдельном классе, который будет использоваться для чтения и записи текущего состояния системы.

19.4 Объединение процедур в работающую программу

Для того, чтобы получить работающую программу, необходимо объединить все приведённые выше процедуры в один файл .cpp. Хотя мы использовали простой тип `float3`, в текущем варианте программу можно компилировать обычным компилятором C++ (например g++), добавив заголовок с описанием простых типов CUDA (`vector_types.h`). Кроме перечисленных процедур, нам потребуется генератор псевдо-случайных числел Ran2

([32] и Приложение 1). Также следует задать необходимые операторы для векторных типов `float3`, операцию нахождения длины вектора (`len(float3 a)`) и его покомпонентного округления (`rint(float3 a)`).

```
int main(int argc, char* argv[]){
  r = (float3*)calloc(N, sizeof(float3));
  v = (float3*)calloc(N, sizeof(float3));
  placeAtoms();
  assignVelocities();
  writeXYZ("O2.coor.xyz", r, N);
  writeXYZ("O2.vel.xyz", v, N);
  return 1;
}
```

В методе `main(...)` необходимо выделить память под координаты и скорости и вызвать процедуры присваивания координат (`placeAtoms()`) и скоростей (`assignVelocities()`) атомов, после чего сохранить эти значения в файлы .xyz (`writeXYZ(...)`). Полученный при запуске программы файл координат можно загрузить в программу VMD и убедиться, что атомы не выходят за пределы куба L×L×L и располагаются достаточно далеко друг от друга (Рис. 43).

20 Свободно движущиеся атомы

20.1 Интегрирование уравнений движения

В данном примере мы будем использовать уравнения движения Ньютона:

$$\begin{aligned} \dot{\vec{v}} &= \vec{a} = \vec{f}/m \\ \dot{\vec{r}} &= \vec{v} \end{aligned} \tag{27}$$

Одним из наиболее популярных алгоритмов интегрирования уравнений движения в молекулярной динамике является алгоритм “прыжок лягушки” (англ. “Leap-Frog”), который можно записать следующим образом:

$$\begin{aligned} \vec{v}(t + \frac{1}{2}\Delta t) &= \vec{v}(t - \frac{1}{2}\Delta t) + \frac{\Delta t}{m}\vec{f}(t) \\ \vec{r}(t + \Delta t) &= \vec{r}(t) + \Delta t \vec{v}(t + \frac{1}{2}\Delta t) \end{aligned} \tag{28}$$

Считая, что скорости частиц заданы между шагами по времени, для одной частицы данный алгоритм в программном коде можно записать следующим образом:

```
1 v += f*(dt/m);
2 r += v*dt;
```

Для системы в N атомов необходимо интегрировать N уравнений движения:

```
1 void integrate(float3* r, float3* v, float3* f,
2           float m, float dt, int N){
3   for(i = 0; i < N; i++){
4     v[i] += f[i]*(dt/m);
5     r[i] += v[i]*dt;
6     f[i] = make_float3(0.0f, 0.0f, 0.0f);
7   }
8 }
```

Отметим, что в данный момент силы, действующие на атомы, равны нулю, так как в нашей программе пока отсутствует реализация потенциалов их взаимодействия. Таким образом, атомы будут свободно перемещаться в пространстве. Обнуление силы после интегрирования уравнений движения производится для того, чтобы сила не накапливалась от одного шага по времени к другому. Кроме того, операции внутри цикла не связаны между собой – на каждом шаге цикла производятся операции только с элементами массива, соответствующими данному шагу цикла (атому). Благодаря этому, параллелизация алгоритма интегрирования достаточно проста – каждому потоку на графической карте должен соответствовать один атом, а адресация массива должна производиться по индексу потока:

```
1  __global__ void integrate_kernel(float3* d_r, float3* d_v,
2            float3* d_f, float m, float dt, int N){
3    int d_i = blockIdx.x*blockDim.x + threadIdx.x;
4    if(d_i < N){
5      float3 r = d_r[d_i];
6      float3 v = d_v[d_i];
7      float3 f = d_f[d_i];
8      v += f*(dt/m);
9      r += v*dt;
10     d_r[d_i] = r;
11     d_v[d_i] = v;
12     d_f[d_i] = make_float3(0.0f, 0.0f, 0.0f);
13   }
14 }
```

Данное вычислительное ядро работает с массивами, имеющими префикс “`d_`” (от англ. “device”). Мы специально используем такой префикс, чтобы отделить указатели на переменные в глобальной памяти ГП от указателей на память ЦП. Далее, мы введём ещё одно правило: если у массива на ЦП есть копия на ГП, мы будем добавлять к такому массиву префикс “`h_`” (от англ. “host”). В нашем случае, этот префикс должны получить массивы `r`, `v` и `f`. В то время, как память под массивы с префиксом “`h_`” выделяется при помощи стан-

дартной процедуры `calloc(...)`, для выделения памяти на ГП необходимо использовать процедуру `cudaMalloc(...)` из диалекта CUDA. Имея два массива "`h_r`" и "`d_r`" одинакового типа (`float3`) и размера (N), можно копировать данные с ЦП на ГП и обратно при помощи процедуры `cudaMemcpy(...)`.

Индексу `d_i` в приведённом листинге присваивается значение глобального индекса потока, по которому и определяется тот элемент массива, с которым работает данный поток. Проверка того, что индекс не выходит за рамки массива (`d_i < N`), делается потому, что количество потоков на ГП кратно размеру блока потоков (то есть может быть больше количества атомов N). Далее, переменные, с которыми данный поток работает, считываются в локальную память потока, с ними производятся все необходимые операции, и результат сохраняется в глобальную память. Такой подход сводит к минимуму количество обращений к глобальной памяти, которые могут существенно замедлить выполнение программы.

20.2 Моделирование свободного движения частиц

Приведённые выше процедуры делают только один шаг интегрирования, в то время как в молекулярной динамике количество этих шагов достигает миллионов или даже миллиардов. Например, если интегрировать с шагов в 1 фс (10^{-15} с), то для достижения 1 нс (10^{-9} с) потребуется 10^6 шагов. Поэтому, процедуру интегрирования необходимо запускать в цикле, количество итераций которого будет определять достигаемый в моделировании временной интервал. Выбор шага интегрирования определяется наиболее быстрыми движениями внутри системы. В нашем случае таковыми будут вибрации ковалентных связей O=O с периодом колебаний порядка 20 фс.

Для того, чтобы следить за динамикой системы, необходимо сохранять координаты атомов в процессе интегрирования уравнений движения. Сохранение координат на каждом шаге не только потребует большого количества места на жёстком диске, но и существенно замедлить процесс вычисления. Поэтому, координаты обычно сохраняют каждые 1000-10000 шагов. Введём переменную `output_freq`, которая будет определять частоту сохранения координат. По истечению `output_freq`, приостановим процесс интегрирования и

сохраним текущие координаты, предварительно перенеся их через границы периодических граничных условий. После сохранения координат, программа должна возвращаться к интегрированию уравнений движения. Этот процесс повторяется до тех пор, пока не будет достигнуто желаемое количество шагов интегрирования `steps_count`:

```
1  void computeCPU(){
2    int s, i;
3    for(step = 0; step <= steps_count; step += output_freq){
4      // Internal cycle - no interaptions
5      for(s = 0; s < output_freq; s++){
6        integrate(h_r, h_v, h_f, m, dt, N);
7      }
8      // Transfer coordinates through periodic boundary
9      for(i = 0; i < N; i++){
10       h_r[i] = transferPBC(h_r[i], L);
11     }
12     // Save coordinates
13     appendXYZ("O2_CPU.coor.xyz", h_r, N);
14   }
15 }
```

В данной вычислительной процедуре мы использовали процедуру `appendXYZ(...)`, которая отличается от `writeXYZ(...)` только модификатором доступа к файлу (вместо “`w`” используется “`a`”), то есть вместо `FILE* file = fopen(filename, "w")` используется `FILE* file = fopen(filename, "a")`. Это позволяет дописывать координаты в файл, сохраняя динамическое поведение системы.

Реализация аналогичной процедуры при интегрировании уравнений движения на ГП выглядит следующим образом:

```
1  void computeGPU(){
2    int s, i;
3    for(step = 0; step <= steps_count; step += output_freq){
```

```
4       // Internal cycle - no interaptions
5       for(s = 0; s < output_freq; s++){
6         integrate_kernel<<<(N-1)/BLOCK_SIZE + 1, BLOCK_SIZE>>>(d_r,
7                   d_v, d_f, m, dt, N);
8       }
9       // Copy coordinates from GPU
10      cudaMemcpy(h_r, d_r, N*sizeof(float3) ,cudaMemcpyDeviceToHost);
11      // Transfer coordinates through periodic boundary
12      for(i = 0; i < N; i++){
13        h_r[i] = transferPBC(h_r[i], L);
14      }
15      // Save coordinates
16      appendXYZ("O2_GPU.coor.xyz", h_r, N);
17    }
18  }
```

Здесь процедура `integrate(...)` заменена на ядро `integrate_kernel<<<...>>>(...)`. Запись `<<<N/BLOCK_SIZE + 1, BLOCK_SIZE>>>` означает, что ядро будет запущено в `N/BLOCK_SIZE + 1` блоках по `BLOCK_SIZE` потоков. Общее число потоков (T) будет определяться размером системы (N), и будет больше либо равно этому размеру ($T \geq N$). Запуск большего числа потоков необходим потому, что потоки на ГП выполняются в блоках, что на аппаратном уровне соответствует разделению потоков между мультипроцессорами. Ядру передаётся тот же набор аргументов, что и процедуре интегрирования на ЦП, но передаются указатели на массивы в памяти ГП. Таким образом, все изменения с данными будут происходить исключительно на ГП и, перед тем как сохранить результаты в файл, необходимо скопировать координаты в память ЦП.

20.3 Объединение процедур в работающую программу

Для запуска моделирования свободного движения несвязанных атомов кислорода необходимо добавить в методе `main(...)` вызов процедуры `computeCPU(...)`. В случае моделирования на ГП память также выделяется и для переменных устройства (`d_r`, `d_v` и `d_f`), в которые копируются данные из памяти ЦП:

```
1 cudaMalloc((void**)&d_r, N*sizeof(float3));
2 cudaMalloc((void**)&d_v, N*sizeof(float3));
3 cudaMalloc((void**)&d_f, N*sizeof(float3));
4 cudaMemcpy(d_r, h_r, N*sizeof(float3), cudaMemcpyHostToDevice);
5 cudaMemcpy(d_v, h_v, N*sizeof(float3), cudaMemcpyHostToDevice);
6 cudaMemcpy(d_f, h_f, N*sizeof(float3), cudaMemcpyHostToDevice);
```

После этого, должна быть вызвана процедура `computeGPU(...)`. Мы нарочно копируем массив с силами, действующими на частицы, так как процедура `cudaMalloc(...)` в отличие от процедуры `calloc(...)` не обнуляет значения массива. В данной программе был использован диалект CUDA, из-за чего скомпилировать её при помощи стандартного компилятора не получится и необходимо использовать компилятор nvcc из CUDA Toolkit. Этот компилятор позволяет также использовать библиотеку векторных операторов `cutil_math.h` из CUDA SDK. После выполнения программы в рабочей директории должен появиться файл .xyz с динамикой системы. Этот файл можно загрузить в программу VMD и проследить движение частиц. Так как атомы кислорода в молекуле находятся на близком расстоянии, VMD догадается, что они связаны валентно и в некоторых представлениях будут прорисованы связи между этими атомами. Не связанные атомы будут двигаться независимо друг от друга, что визуально отразится в нефизическом растяжении связей. Это будет исправлено в следующем разделе, где мы добавим потенциал ковалентного взаимодействия.

21 Потенциал валентных взаимодействий

21.1 Формулы для расчёта сил

Реализация любого потенциала начинается с его формальной записи. В частности, чтобы получить формулу для расчёта силы, действующей на атом, необходимо вычислить производную потенциала по координатам этого атома. В случае молекулы O_2, каждый атом связан валентно только с одним другим атомом – его соседом по молекуле. Таким образом, от координат каждого атома будет зависеть только один член суммы валентных потенциалов V_{bond} из уравнения 24:

$$V_{bond}^{i} = \frac{1}{2} K_s (r_{ij} - b_0)^2 \tag{29}$$

Введём оператор градиента по координатам атома i:

$$\nabla_i = \begin{pmatrix} \frac{\partial}{\partial x_i} \\ \frac{\partial}{\partial y_i} \\ \frac{\partial}{\partial z_i} \end{pmatrix}. \tag{30}$$

Тогда:

$$\begin{aligned} \nabla_i r_{ij} &= \nabla_i \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2 + (z_j - z_i)^2} = \\ &= \frac{1}{2r_{ij}} \nabla_i \left((x_j - x_i)^2 + (y_j - y_i)^2 + (z_j - z_i)^2 \right) = \\ &= \frac{1}{2r_{ij}} \begin{pmatrix} \frac{\partial (x_j - x_i)^2}{\partial x_i} \\ \frac{\partial (y_j - y_i)^2}{\partial y_i} \\ \frac{\partial (z_j - z_i)^2}{\partial z_i} \end{pmatrix} = \frac{1}{2r_{ij}} \begin{pmatrix} -2(x_j - x_i) \\ -2(y_j - y_i) \\ -2(z_j - z_i) \end{pmatrix} = -\frac{\vec{r}_{ij}}{r_{ij}}, \end{aligned} \tag{31}$$

где $\vec{r}_{ij} = \vec{r}_j - \vec{r}_i$. Используя данное выражение, можно найти градиент потенциала ковалентной связи:

$$\nabla_i V^i_{bond} = \frac{\partial V^i_{bond}}{\partial r_{ij}} \nabla_i r_{ij} = -K_s(r_{ij} - b_0)\frac{\vec{r}_{ij}}{r_{ij}} \tag{32}$$

Таким образом, сила, действующая на атом i из-за ковалентного взаимодействия с атомом j будет задаваться следующим выражением:

$$\vec{f}^{ij}_{bond} = -\nabla_i V^i_{bond} = K_s(r_{ij} - b_0)\frac{\vec{r}_{ij}}{r_{ij}} = -\vec{f}^{ji}_{bond} \tag{33}$$

Так как в случае молекул O=O каждый атом образует только одну связь, сила $\vec{f}^{ij}_{bond}$ будет полной силой ковалентного взаимодействия для атома i, а $\vec{f}^{ji}_{bond}$ – для атома j, то есть $\vec{f}^{i}_{bond} = \vec{f}^{ij}_{bond}$ и $\vec{f}^{j}_{bond} = \vec{f}^{ji}_{bond}$. В общем случае, сила $\vec{f}^{i}_{bond}$ будет являться суммой сил $\vec{f}^{ij}_{bond}$, посчитанных для всех ковалентных связей атома i, а параметры взаимодействий K_s и b_0 будут зависеть от типов атомов i и j, то есть:

$$\vec{f}^{i}_{bond} = \sum_{j \in (j_1,\ldots,j_B)} \vec{f}^{ij}_{bond} = \sum_{j \in (j_1,\ldots,j_B)} K^{ij}_s(r_{ij} - b^{ij}_0)\frac{\vec{r}_{ij}}{r_{ij}}, \tag{34}$$

где $(j_1, \ldots, j_B)$ – набор атомов, ковалентно связанных с атомом i.

21.2 Программная реализация расчёта потенциала

Как было обговорено ранее, для рассматриваемой системы ковалентные взаимодействия связывают атомы с чётными индексами ($N = 0, 2, \ldots, N-2$) с атомами, у которых индекс на единицу больше ($N = 1, 3, \ldots, N-1$). Таким образом, расчёт силы, действующей между двумя атомами, можно записать в следующем виде:

```
1 void computeCovalent(float3* h_r, float3* h_f,
2           float Ks, float b0, int N, float L){
3   int i;
4   float3 rij;
```

```
5   for(i = 0; i < N; i+=2){
6     rij = getVector(h_r[i], h_r[i+1], L);
7     float b = len(rij);
8     float df = Ks*(b - b0)/b;
9     float3 fij = df*rij;
10    h_f[i] += fij;
11    h_f[i+1] -= fij;
12  }
13 }
```

Рассчитанная в данной процедуре сила зависит от координат и используется в интеграторе, который эти координаты изменяет на каждом шаге интегрирования по времени. Таким образом, на каждом временном шаге силу необходимо рассчитывать заново, что отражается в следующем изменении процедуры `computeCPU()`:

```
1  void computeCPU(){
2    ...
3      for(s = 0; s < output_freq; s++){
4        computeCovalent(h_r, h_f, Ks, b0, N, L);
5        integrate(h_r, h_v, h_f, m, dt, N);
6      }
7    ...
8  }
```

Так как каждый атом имеет только одну ковалентную связь, суммирование сил на ГП можно производить не опасаясь того, что два потока будут одновременно сохранять результат в одну ячейку памяти. Количество потоков на ГП при этом будет равно количеству ковалентных связей (то есть $N/2$). Кроме того, поскольку данный потенциал считается первым, возможно прямое сохранение рассчитанной в данном ядре силы в соответствующую ячейку массива сил, действующих на атомы системы (вместо прибавления соответствующих значений).

```
__global__ void computeCovalent_kernel(float3* d_r, float3* d_f,
          float Ks, float b0, int N, float L){
  int d_i = blockIdx.x*blockDim.x + threadIdx.x;
  if(d_i < N/2){
    float3 ri = d_r[2*d_i];
    float3 rj = d_r[2*d_i + 1];
    float3 rij = getVector(ri, rj, L);
    float b = len(rij);
    float df = Ks*(b - b0)/b;
    float3 fij = df*rij;
    d_f[2*d_i] = fij;
    d_f[2*d_i + 1] = -fij;
  }
}
```

В данном ядре значения координат взаимодействующих частиц сначала сохраняются в локальную память потока и затем используются в расчёте сил взаимодействия. Полученная сила сохраняется для двух атомов с противоположными знаками. Изменения в общей процедуре расчёта на ГП аналогичны изменениям кода на ЦП:

```
void computeGPU(){
  ...
    for(s = 0; s < output_freq; s++){
      computeCovalent_kernel
          <<<(N/2-1)/BLOCK_SIZE + 1, BLOCK_SIZE>>>(d_r,
                d_f, Ks, b0, N, L);
      integrate_kernel<<<(N-1)/BLOCK_SIZE + 1, BLOCK_SIZE>>>(d_r,
                d_v, d_f, m, dt, N);
    }
  ...
}
```

В результате выполнения полученной программы молекулы кислорода будут свободно двигаться, а атомы внутри каждой молекулы – колебаться вокруг равновесного расстояния. Следующим шагом станет добавление взаимодействий между молекулами кислорода.

22 Невалентные взаимодействия Ван-дер-Ваальса

22.1 Формальная запись

Взаимодействия Ван-дер-Ваальса, которые включают в себя сильное отталкивание атомов на коротких дистанциях (отталкивания Паули) и слабое притяжение на длинных (дисперсионные силы), описываются потенциалом Леннарда-Джонса:

$$V_{LJ} = \sum_{i,j} \varepsilon \left[\left(\frac{\sigma}{r_{ij}} \right)^{12} - 2 \left(\frac{\sigma}{r_{ij}} \right)^{6} \right], \tag{35}$$

Суммирование обычно производится по всем возможным парам (i, j), кроме тех пар, которые связаны ковалентно или образуют ковалентный угол. В случае кислорода из этой суммы нужно исключить взаимодействия между атомами одной молекулы. Также следует исключить двойной расчёт потенциала – если сила была рассчитана для пары i-j, то её не следует считать для пары j-i. Потенциал взаимодействия между атомами i и j можно записать следующим образом:

$$V_{LJ}^{ij} = \varepsilon \left[\left(\frac{\sigma}{r_{ij}} \right)^{12} - 2 \left(\frac{\sigma}{r_{ij}} \right)^{6} \right] \tag{36}$$

Сила, действующая на атом i, будет определяться суммой всех бинарных потенциалов, в которых задействован атом i, то есть:

$$\begin{aligned} \vec{f_i} &= -\sum_{j=0}^{N-1} \nabla_i V_{LJ}^{ij} = -\sum_{j=0}^{N-1} \frac{\partial V_{LJ}^{ij}}{\partial r_{ij}} \nabla_i r_{ij} = -\sum_{j=0}^{N-1} \varepsilon \left[-12 \frac{\sigma^{12}}{r_{ij}^{13}} + 12 \frac{\sigma^{6}}{r_{ij}^{7}} \right] \left(-\frac{\vec{r}_{ij}}{r_{ij}} \right) = \\ &= -\sum_{j=0}^{N-1} 12\varepsilon \left[\left(\frac{\sigma}{r_{ij}} \right)^{12} - \left(\frac{\sigma}{r_{ij}} \right)^{6} \right] \frac{\vec{r}_{ij}}{r_{ij}^2} = \sum_{j=0}^{N-1} 12\varepsilon \left(\frac{\sigma}{r_{ij}} \right)^{6} \left[1 - \left(\frac{\sigma}{r_{ij}} \right)^{6} \right] \frac{\vec{r}_{ij}}{r_{ij}^2} \end{aligned} \tag{37}$$

22.2 Программная реализация

На центральном процессоре расчёт потенциала Леннарда-Джонса можно разложить в два цикла. Первый цикл пробегает по всем атомам i, второй – по всем атомам j так, что

каждая пара i-j считается только один раз. Кроме того, необходимо добавить проверку на то, что частицы i и j не связаны ковалентно. Посчитанная сила взаимодействия между атомами i и j добавляется к общей силе, действующей на эти атомы, с разными знаками.

```
1  void computeLJ(float3* h_r, float3* h_f,
2          float epsilon, float sigma, int N, float L){
3    int i, j;
4    float3 rij;
5    for(i = 0; i < N; i++){
6      for(j = i+1; j < N; j++){
7        if(!(j == i+1 && i%2 == 0)){
8          rij = getVector(h_r[i], h_r[j], L);
9          float rij2 = len2(rij);
10         float rij2inv = 1.0f/rij2;
11         float sor2 = sigma*sigma*rij2inv;
12         float sor6 = sor2*sor2*sor2;
13         float df = 12.0f*epsilon*sor6*(1.0f - sor6)*rij2inv;
14         float3 fij = df*rij;
15         h_f[i] += fij;
16         h_f[j] -= fij;
17       }
18     }
19   }
20 }
```

До сих пор реализация процедур на ГП слабо отличалась от аналога на ЦП. В случае невалентных взаимодействий эти различия становятся более существенными. Связано это главным образом с тем, что в параллельной реализации необходимо уделять внимание записи и чтению данных разными потоками. Например, если два потока i и j сохраняют результат вычислений в одну и ту же ячейку памяти, может оказаться что они делают это одновременно. В этом случае невозможно сказать, какой результат из двух сохранённых

окажется в этой ячейке по окончанию работы программы. Существует несколько возможных способов решения данной проблемы.

В NVidia CUDA SDK есть хороший пример, в котором рассматриваются взаимодействия тел через гравитационный потенциал. Хотя в данном примере приведена очень эффективная реализация схожей задачи на ГП, для реализации потенциала Леннарда-Джонса в молекулярных системах она не очень хорошо подходит. Дело в том, что потенциал Леннарда-Джонса убывает гораздо быстрее ($\sim r^{-6}$) гравитационного или электростатического потенциала ($\sim r^{-1}$). Это позволяет ввести в расчёте радиус обрезки, считая нулевым потенциал (и силу взаимодействия) между частицами, находящимися друг от друга дальше этого радиуса. Далее, можно составлять список соседей – атомов, которые либо находятся внутри радиуса обрезки, либо могут в ближайшее время в нём оказаться (иногда также называют списком Верле). В зависимости от скорости частиц (температуры), "запаса" заданного при составлении списка соседей и прочих параметров моделирования, можно задать частоту обновления списка в шагах по времени. Таким образом, составлять данный список, а, следовательно, считать расстояния между всеми возможными парами атомов, нужно будет не на каждом шагу, так как при расчёте потенциала будут рассматриваться только пары атомов из списка соседей. Это позволяет существенно ускорить процесс вычислений, в особенности для больших систем. К сожалению, предложенный в CUDA SDK алгоритм расчёта потенциала взаимодействия в задаче N-тел невозможно дополнить списком Верле.

Другим решением проблемы синхронизации доступа к памяти является разделение расчёта на два вычислительных ядра, одно из которых считает силы взаимодействия, сохраняя их в уникальные ячейки памяти. Второе ядро при этом занимается исключительно суммированием полученных величин для всех частиц в системе. Этот подход сложен в реализации (особенно, при составлении списков соседей), а также имеет преимущество в производительности только при существенной арифметической сложности расчёта потенциала.

Наиболее популярным методом расчёта сил взаимодействия между атомами является подход, в котором одному потоку ставится в соответствие одна частица. При этом рас-

считанная сила сохраняется только для частицы соответствующей потоку, то есть сила взаимодействия между частицами i и j рассчитывается дважды – в потоках i и j. С одной стороны, двойной пересчёт удваивает количество вычислений, с другой стороны простота этого подхода и минимальное число обращений к глобальной памяти ГП позволяют достичь достаточно хорошей производительности. Как и раньше, все величины загружаются из глобальной памяти в локальную, где с ними производятся необходимые операции, результат которых сохраняется в глобальную память.

```
__global__ void computeLJ_kernel(float3* d_r, float3* d_f,
        float epsilon, float sigma, int N, float L){
  int d_i = blockIdx.x*blockDim.x + threadIdx.x;
  if(d_i < N){
    int j;
    float3 fi = d_f[d_i];
    float3 ri = d_r[d_i];
    for(j = 0; j < N; j++){
      if(!(d_i == j || (j == d_i+1 && d_i%2 == 0) ||
        (d_i == j+1 && j%2 == 0))){
        float3 rj = d_r[j];
        float3 rij = getVector(ri, rj, L);
        float rij2 = len2(rij);
        float rij2inv = 1.0f/rij2;
        float sor2 = sigma*sigma*rij2inv;
        float sor6 = sor2*sor2*sor2;
        float df = 12.0f*epsilon*sor6*(1.0f - sor6)*rij2inv;
        float3 fij = df*rij;
        fi += fij;
      }
    }
    d_f[d_i] = fi;
```

```
23    }
24  }
```

Кроме проверки на ковалентную связь между атомами i и j добавилась проверка на идентичность (`d_i != j`). Координаты частицы i, а также сила, действующая на эту частицу считываются из глобальной памяти в самом начале вычислительного ядра. Значение силы до запуска этого ядра содержит в себе силу, действующую из-за ковалентного потенциала. После вычисления всех бинарных взаимодействий между атомом i и остальными атомами системы, сила сохраняется в глобальную память.

22.3 Списки соседей

Расчёт невалентных взаимодействий является наиболее затратной процедурой в молекулярных симуляциях. Чтобы ускорить вычислительный процесс, добавим в программу списки соседей, в которые будут входить частицы, находящиеся на расстоянии менее чем `pairs_cutoff` друг от друга. При этом сам расчёт будем вести только для частиц, которые находятся на расстоянии менее, чем `lj_cutoff`. Кроме того, из списка должны быть исключены пары связанных ковалентно атомов, и пары i-i. Для этого создадим процедуру, которая будет возвращать `true`, если атомы не должны быть добавлены в список и `false` в обратном случае:

```
1  inline __host__ __device__ bool checkExcluded(int i, int j){
2    if(i == j){
3      return true;
4    }
5    if(j == i+1 && i%2 == 0){
6      return true;
7    }
8    if(i == j+1 && j%2 == 0){
9      return true;
10   }
```

```
  return false;
}
```

Список соседей будет представлять собой массив переменных типа `int2`, два целочисленных элемента которых соответствуют индексам атомов i и j, добавленным в список. Кроме того, так как количество соседей будет зависеть от взаимного расположения атомов в конкретный момент времени, необходимо также сохранять количество пар в списке:

```
void generatePairslist(float3* h_r, int pairs_cutoff2, float L,
          int2* h_pairslist, int* pairsCount){
  *pairsCount = 0;
  int i, j;
  for(i = 0; i < N; i++){
    for(j = i+1; j < N; j++){
      if(!checkExcluded(i, j)){
        float3 dr = getVector(h_r[i], h_r[j], L);
        if(len2(dr) < pairs_cutoff2){
          h_pairslist[*pairsCount].x = i;
          h_pairslist[*pairsCount].y = j;
          (*pairsCount)++;
        }
      }
    }
  }
}
```

Здесь `pairs_cutoff2` обозначает квадрат радиуса обрезки, что позволяет избавиться от необходимости вычисления квадратного корня из расстояния между атомами. Для расчёта потенциала Леннарда-Джонса с использованием списка соседей необходимо в цикле рассмотреть все пары атомов из списка и рассчитать потенциал для тех, атомы в которых находятся на расстоянии меньше `lj_cutoff` друг от друга:

```
void computeLJWithPairslist(float3* h_r, float3* h_f,
```

```
2           int2* h_pairslist, int pairsCount, float lj_cutoff2,
3           float epsilon, float sigma, int N, float L){
4   int p;
5   float3 rij;
6   for(p = 0; p < pairsCount; p++){
7     int2 pair = h_pairslist[p];
8     int i = pair.x;
9     int j = pair.y;
10    rij = getVector(h_r[i], h_r[j], L);
11    float rij2 = len2(rij);
12    if(rij2 < lj_cutoff2){
13      float rij2inv = 1.0f/rij2;
14      float sor2 = sigma*sigma*rij2inv;
15      float sor6 = sor2*sor2*sor2;
16      float df = 12.0f*epsilon*sor6*(1.0f - sor6)*rij2inv;
17      float3 fij = df*rij;
18      h_f[i] += fij;
19      h_f[j] -= fij;
20    }
21  }
22 }
```

С учётом необходимости составления списка соседей, общая процедура вычислений будет несколько изменена: частота выхода из внутреннего цикла будет определяться наибольшим общим делителем частоты сохранения координат и частоты составления списка соседей (процедура `GCD(...)` – англ. Greatest Common Divisor):

```
1 void computeCPU(){
2   int s, i;
3   int stride = GCD(pairs_freq, output_freq);
4   for(step = 0; step <= steps_count; step += stride){
```

```
5       if(step % pairs_freq == 0){
6         generatePairslist(h_r,
7             pairs_cutoff*pairs_cutoff, L, h_pairslist, &pairs_count);
8       }
9       for(s = 0; s < stride; s++){
10        computeCovalent(h_r, h_f, Ks, b0, N, L);
11        computeLJWithPairslist(h_r, h_f,
12            h_pairslist, pairs_count,
13            lj_cutoff*lj_cutoff, epsilon, sigma, N, L);
14        integrate(h_r, h_v, h_f, m, dt, N);
15      }
16      if(step % output_freq == 0){
17        for(i = 0; i < N; i++){
18          h_r[i] = transferPBC(h_r[i], L);
19        }
20        appendXYZ("O2_CPU.coor.xyz", h_r, N);
21      }
22    }
23  }
```

Так как на ГП каждому потоку ставится в соответствие одна частица, список соседей должен быть организован иначе. Во-первых он будет состоять из переменных типа `int`, значение которых будет определять индекс атома, с которым атом, соответствующий потоку взаимодействует. Во-вторых, этот массив будет двухмерным – одно измерение будет определять частицу i, которая ставится в соответствие потоку, а в другом измерении будут храниться все индексы частиц j, с которыми частица i взаимодействует.

```
1  __global__ void generatePairslist_kernel(float3* d_r,
2            float pairs_cutoff2, int N, float L,
3            int* d_pairs_count, int* d_pairslist){
4    int d_i = blockIdx.x*blockDim.x + threadIdx.x;
```

```
5   if(d_i < N){
6     int j;
7     float3 ri = d_r[d_i];
8     int pairs_count = 0;
9     for(j = 0; j < N; j++){
10      if(!checkExcluded(d_i, j)){
11        float3 rj = d_r[j];
12        float3 rij = getVector(ri, rj, L);
13        float rij2 = len2(rij);
14        if(rij2 < pairs_cutoff2){
15          d_pairslist[pairs_count*N + d_i] = j;
16          pairs_count ++;
17        }
18      }
19    }
20    d_pairs_count[d_i] = pairs_count;
21  }
22 }
```

Как видно из данной процедуры, доступ к двумерному массиву осуществляется как к одномерному. При этом номер рассматриваемой пары умножается на длину строки массива. Такой способ адресации позволяет явно контролировать расположение данных в памяти, делая этот доступ последовательным (потоки i и $i+1$ читают одновременно данные из двух последовательных ячеек массива). При работе с глобальной памятью такой шаблон доступа является наиболее эффективным с точки зрения производительности. Ядро сохраняет все найденные пары в список и считает общее их количество для каждой частицы. Массивы со списком пар и количеством пар для частиц могут быть использованы в вычислительном ядре невалентных взаимодействий:

```
1 __global__ void computeLJWithPairslist_kernel(float3* d_r, float3* d_f,
2           int* d_pairs_count, int* d_pairslist, float lj_cutoff2,
```

```
3           float epsilon, float sigma, int N, float L){
4   int d_i = blockIdx.x*blockDim.x + threadIdx.x;
5   if(d_i < N){
6     int p;
7     float3 rij;
8     float3 fi = d_f[d_i];
9     float3 ri = d_r[d_i];
10    for(p = 0; p < d_pairs_count[d_i]; p++){
11      int j = d_pairslist[p*N + d_i];
12      float3 rj = d_r[j];
13      rij = getVector(ri, rj, L);
14      float rij2 = len2(rij);
15      if(rij2 < lj_cutoff2){
16        float rij2inv = 1.0f/rij2;
17        float sor2 = sigma*sigma*rij2inv;
18        float sor6 = sor2*sor2*sor2;
19        float df = 12.0f*epsilon*sor6*(1.0f - sor6)*rij2inv;
20        float3 fij = df*rij;
21        fi += fij;
22      }
23    }
24    d_f[d_i] = fi;
25  }
26 }
```

Как видно из данной программной реализации, цикл внутри ядра теперь пробегает только по парам из списка соседей, что существенно уменьшает необходимое количество вычислений. Кроме того, проверка на исключение из списка для атомов, связанных ковалентно, теперь делается при составлении списка соседей, то есть не на каждом шаге интегрирования.

Изменения в основной части программы аналогичны версии на ЦП:

```
void computeGPU(){
  int s, i;
  int stride = GCD(pairs_freq, output_freq);
  for(step = 0; step <= steps_count; step += stride){
    if(step % pairs_freq == 0){
      generatePairslist_kernel
          <<<(N-1)/BLOCK_SIZE + 1, BLOCK_SIZE>>>(d_r,
            pairs_cutoff*pairs_cutoff, N, L,
            d_pairs_count, d_pairslist);
    }
    for(s = 0; s < stride; s++){
      computeCovalent_kernel
          <<<(N/2-1)/BLOCK_SIZE + 1, BLOCK_SIZE>>>(d_r,
            d_f, Ks, b0, N, L);
      computeLJWithPairslist_kernel
          <<<(N-1)/BLOCK_SIZE + 1, BLOCK_SIZE>>>(d_r,
            d_f, d_pairs_count, d_pairslist, lj_cutoff*lj_cutoff,
            epsilon, sigma, N, L);
      integrate_kernel<<<(N-1)/BLOCK_SIZE + 1, BLOCK_SIZE>>>(d_r,
          d_v, d_f, m, dt, N);
    }
    if(step % output_freq == 0){
      cudaMemcpy(h_r, d_r, N*sizeof(float3) ,cudaMemcpyDeviceToHost);
      for(i = 0; i < N; i++){
        h_r[i] = transferPBC(h_r[i], L);
      }
      cudaMemcpy(d_r, h_r, N*sizeof(float3), cudaMemcpyHostToDevice);
      appendXYZ("O2_GPU.coor.xyz", h_r, N);
    }
```

```
30     }
31 }
```

23 Расчёт термодинамических величин

23.1 Энергия системы

Силы, действующие на атомы системы были рассчитаны через производную от энергии для каждого бинарного взаимодействия. Таким образом, расчёт энергии достаточно тривиален – с теми же параметрами и для тех же пар атомов необходимо посчитать исходную величину согласно уравнению 24. При этом, можно разделить потенциальную энергию по компонентам, выводя отдельно энергию ковалентных и нековалентных взаимодействий, а также – их сумму (полную потенциальную энергию). В программном коде на ЦП добавление такой функциональности тривиально. Например, в расчёте потенциала ковалентных связей необходимо добавить следующее:

```
1   ...
2     float df = Ks*(b - b0)/b;
3     pot_cov += 0.5f*Ks*(b - b0)*(b - b0);
4     float3 fij = df*rij;
5   ...
6 }
```

Переменная `pot_cov` аккумулирует потенциальную энергию ковалентных взаимодействий на каждом шаге интегрирования. Таким образом, в ней накапливается суммарная энергия и чтобы найти её среднее значение, необходимо разделить полученную величину на количество шагов по времени на протяжении которого она аккумулировалась. Вывод этой величины можно производить одновременно с сохранением координат в файл xyz. После вывода значение следует обнулить. Аналогичную процедуру добавим и в расчёт потенциала Леннарда-Джонса, а вывод этих значений на экран выглядит следующим образом:

```
1 ...
2     if(step % output_freq == 0){
3       for(i = 0; i < N; i++){
4         h_r[i] = transferPBC(h_r[i], L);
```

```
5       }
6       appendXYZ("O2_CPU.coor.xyz", h_r, N);
7       pot_cov /= output_freq;
8       pot_lj /= output_freq;
9       printf("%ld\t%f\t%f\t%f\n",
10            step, pot_cov, pot_lj, pot_cov + pot_lj);
11      pot_cov = 0.0;
12      pot_lj = 0.0;
13    }
14 ...
```

В первую колонку выводится шаг по времени, во вторую и третью колонки – значения энергии ковалентных и нековалентных взаимодействий. В последнюю колонку выводится полная энергия системы.

На графическом процессоре, потоки изолированны друг от друга. Поэтому, расчёт энергии удобнее производить для каждого атома в отдельности, суммируя конечный результат непосредственно перед выводом. Хотя последнее можно осуществлять и на ГП при помощи эффективных алгоритмов редукции [33, 34], здесь мы будем копировать весь массив данных, находя сумму уже на ЦП. Для хранения промежуточных значений энергии нам понадобится одномерный массив чисел с плавающей точкой размером N в памяти ГП, а для нахождения суммы – его аналог в памяти ЦП. Скопировав результат из памяти ГП в память ЦП, найдём сумму энергий по всем атомам и обнулим значения в памяти ГП:

```
1 ...
2     if(step % output_freq == 0){
3
4       cudaMemcpy(h_r, d_r, N*sizeof(float3), cudaMemcpyDeviceToHost);
5       cudaMemcpy(h_pot_cov, d_pot_cov, N*sizeof(float),
6         cudaMemcpyDeviceToHost);
7       cudaMemcpy(h_pot_lj, d_pot_lj, N*sizeof(float),
8         cudaMemcpyDeviceToHost);
```

```
9
10        pot_cov = 0.0;
11        pot_lj = 0.0;
12        for(i = 0; i < N; i++){
13          h_r[i] = transferPBC(h_r[i], L);
14          pot_cov += h_pot_cov[i];
15          pot_lj += h_pot_lj[i];
16        }
17        pot_cov /= output_freq;
18        pot_lj /= 2*output_freq;
19
20        cudaMemcpy(d_r, h_r, N*sizeof(float3), cudaMemcpyHostToDevice);
21        cudaMemset(d_pot_cov, 0, N*sizeof(float));
22        cudaMemset(d_pot_lj, 0, N*sizeof(float));
23
24        appendXYZ("O2_GPU.coor.xyz", h_r, N);
25        printf("%ld\t%f\t%f\t%f\n",
26          step, pot_cov, pot_lj, pot_cov + pot_lj);
27      }
28  ...
```

Потенциальная энергия взаимодействия Леннарда-Джонса делится пополам, так как одно и тоже бинарное взаимодействие между атомами i и j рассчитывается дважды – в i-том и в j-том потоках. Расчёт потенциальной энергии является тривиальным дополнением ядра, вычисляющего межатомные силы для этого взаимодействия. Например, для потенциала Леннарда-Джонса дополненное ядро принимает следующий вид:

```
1  __global__ void computeLJWithPairslist_kernel(float3* d_r, float3* d_f,
2            float* d_pot_lj,
3            int* d_pairs_count, int* d_pairslist, float lj_cutoff2,
4            float epsilon, float sigma, int N, float L){
```

```
5   int d_i = blockIdx.x*blockDim.x + threadIdx.x;
6   if(d_i < N){
7     int p;
8     float3 rij;
9     float3 fi = d_f[d_i];
10    float3 ri = d_r[d_i];
11    float pot = d_pot_lj[d_i];
12    for(p = 0; p < d_pairs_count[d_i]; p++){
13      int j = d_pairslist[p*N + d_i];
14      float3 rj = d_r[j];
15      rij = getVector(ri, rj, L);
16      float rij2 = len2(rij);
17      if(rij2 < lj_cutoff2){
18        float rij2inv = 1.0f/rij2;
19        float sor2 = sigma*sigma*rij2inv;
20        float sor6 = sor2*sor2*sor2;
21        pot += epsilon*sor6*(sor6 - 2.0f);
22        float df = 12.0f*epsilon*sor6*(1.0f - sor6)*rij2inv;
23        float3 fij = df*rij;
24        fi += fij;
25      }
26    }
27    d_f[d_i] = fi;
28    d_pot_lj[d_i] = pot;
29  }
30 }
```

Потенциальная энергия Леннарда-Джонса на ГП аккумулируется отдельно для каждого атома в массиве `d_pot_lj`. Значение соответствующее потоку считывается из этого массива в начале выполнения ядра, а обновляется и сохраняется по окончанию выполнения ядра.

23.2 Средняя температура

Начальная температура системы задаётся через распределение Максвела-Больцмана, а расчёт средней температуры – процесс обратный её заданию. Как и компоненты скоростей, кинетические энергии распределены нормально, а их среднее будет определять текущую температуру. В процессе вычисления, величины кинетических энергий имеет смысл округлять на протяжении некоторого количества шагов интегрирования. Особенно это актуально для систем, состоящих из небольшого числа атомов. Таким образом, необходимо сохранять сумму кинетических энергий все атомов (в нашем примере – квадратов их скоростей). Обычно это делается в интеграторе уравнений движения, так как именно там скорость атомов приращивается согласно рассчитанному значению силы. Отметим, что в интеграторе "прыжок лягушки" скорости рассчитываются на полушаге. Использование этих скоростей ведёт к несколько завышенному значению температуры. Чтобы получить значение скоростей на целом значении шага, можно разбить приращение скорости на два полуприращения, используя промежуточное значение при расчёте температуры. Изменённый интегратор при этом выглядит следующим образом:

```
1  void integrate(float3* h_r, float3* h_v, float3* h_f,
2          float m, float dt, int N){
3    int i;
4    for(i = 0; i < N; i++){
5      float3 halfdv = 0.5f*h_f[i]*(dt/m);
6      h_v[i] += halfdv;
7      temperature += len2(h_v[i]);
8      h_v[i] += halfdv;
9      h_r[i] += h_v[i]*dt;
10     h_f[i] = make_float3(0.0f, 0.0f, 0.0f);
11   }
12 }
```

Чтобы из полной кинетической энергии получить температуру, необходимо найти наиболее вероятную (в данном случае – среднюю) кинетическую энергию атома и разделить на постоянную Больцмана:

```
...
    if(step % output_freq == 0){
      for(i = 0; i < N; i++){
        h_r[i] = transferPBC(h_r[i], L);
      }
      appendXYZ("O2_CPU.coor.xyz", h_r, N);
      pot_cov /= output_freq;
      pot_lj /= output_freq;
      temperature *= 0.5*m/N;
      temperature /= output_freq*kB;
      printf("%ld\t%f\t%f\t%f\t%f\n",
            step, pot_cov, pot_lj, pot_cov + pot_lj, temperature);
      pot_cov = 0.0;
      pot_lj = 0.0;
      temperature = 0.0;
    }
...
```

Полученные конечные значения температуры могут несколько отличаться от изначально заданных 300K. Это связано с тем, что расположения атомов в системе не идеально – в начальной конформации присутствуют излишки потенциальной энергии. Это можно исправить проведением эквилибрации, в процессе которой скорости атомов периодически переназначаются.

На ГП, расчёт температуры также осуществляется в интеграторе, но значения сохраняются для каждого потока (атома) отдельно:

```
__global__ void integrate_kernel(float3* d_r, float3* d_v, float3* d_f,
          float* d_temperature, float m, float dt, int N){
```

```
3   int d_i = blockIdx.x*blockDim.x + threadIdx.x;
4   if(d_i < N){
5     float3 r = d_r[d_i];
6     float3 v = d_v[d_i];
7     float3 f = d_f[d_i];
8     float3 halfdv = 0.5f*f*(dt/m);
9     v += halfdv;
10    d_temperature[d_i] += len2(v);
11    v += halfdv;
12    r += v*dt;
13    d_r[d_i] = r;
14    d_v[d_i] = v;
15    d_f[d_i] = make_float3(0.0f, 0.0f, 0.0f);
16  }
17 }
```

Где `d_temperature` – массив переменных типа `float`, выделенный в памяти ГП. Затем, как и для энергии, полученные значения сумм квадратов температур копируются в память ЦП, где и ведётся расчёт средней кинетической энергии и температуры:

```
1  ...
2     if(step % output_freq == 0){
3       cudaMemcpy(h_r, d_r, N*sizeof(float3), cudaMemcpyDeviceToHost);
4       cudaMemcpy(h_pot_cov, d_pot_cov, N*sizeof(float,
5             cudaMemcpyDeviceToHost);
6       cudaMemcpy(h_pot_lj, d_pot_lj, N*sizeof(float),
7             cudaMemcpyDeviceToHost);
8       cudaMemcpy(h_temperature, d_temperature, N*sizeof(float),
9             cudaMemcpyDeviceToHost);
10      pot_cov = 0.0;
11      pot_lj = 0.0;
```

```
12        temperature = 0.0;
13        for(i = 0; i < N; i++){
14          h_r[i] = transferPBC(h_r[i], L);
15          pot_cov += h_pot_cov[i];
16          pot_lj += h_pot_lj[i];
17          temperature += h_temperature[i];
18        }
19        temperature *= 0.5*m/N;
20        temperature /= output_freq*kB;
21        pot_cov /= output_freq;
22        pot_lj /= 2*output_freq;
23        cudaMemcpy(d_r, h_r, N*sizeof(float3), cudaMemcpyHostToDevice);
24        cudaMemset(d_pot_cov, 0, N*sizeof(float));
25        cudaMemset(d_pot_lj, 0, N*sizeof(float));
26        cudaMemset(d_temperature, 0, N*sizeof(float));
27        appendXYZ("O2_GPU.coor.xyz", h_r, N);
28        printf("%ld\t%f\t%f\t%f\t%f\n",
29              step, pot_cov, pot_lj, pot_cov + pot_lj, temperature);
30      }
31  ...
```

Здесь также можно использовать алгоритмы редукции в памяти ГП и копировать уже конечное значение температуры.

Приложение 1 Реализация генератора псевдо-случайных чисел Ran2

```
1  #define IM1   2147483563
2  #define IM2   2147483399
3  #define AM    (1.0/IM1)
4  #define IMM1  (IM1 -1)
5  #define IA1   40014
6  #define IA2   40692
7  #define IQ1   53668
8  #define IQ2   52774
9  #define IR1   12211
10 #define IR2   3791
11 #define NTAB  32
12 #define NDIV  (1 + IMM1/NTAB)
13 #define EPS1   1.2e-7
14 #define RNMX  (1.0 - EPS1)
15
16 inline double ran2(int *idum){
17   int j;
18   int k;
19   static int  idum2 = 123456789;
20   static int  iy = 0;
21   static int  iv[NTAB];
22   double      tempran;
23   if(*idum <= 0){   /* initialize */
24     if(-(*idum) < 1){
25       *idum = 1; /* be sure to prevent idum = 0 */
26     } else {
27       *idum = - (*idum);
```

```
28      }
29      idum2 = (*idum);
30      for(j = NTAB+7; j >= 0; j--){ /*load shuffle table (after 8 warm-ups) */
31        k = (*idum)/IQ1;
32        *idum = IA1*(*idum - k*IQ1) - k*IR1;
33        if(*idum < 0){
34          *idum += IM1;
35        }
36        if(j < NTAB){
37          iv[j] = *idum;
38        }
39      }
40      iy = iv[0];
41    }
42    k = (*idum)/IQ1;                    /* start here when not init. */
43    *idum = IA1*(*idum - k*IQ1) - k*IR1;
44    if(*idum < 0){
45      *idum += IM1;
46    }
47    k = idum2/IQ2;
48    idum2 = IA2*(idum2 - k*IQ2) - k*IR2;
49    if(idum2 < 0){
50      idum2 += IM2;
51    }
52    j = iy/NDIV;
53    iy = iv[j] - idum2;
54    iv[j] = *idum;
55    if(iy < 1){
56      iy += IMM1;
57    }
```

```
58    if((tempran = AM*iy) > RNMX){
59      return RNMX;
60    }
61    else return tempran;
62  }
```

Список литературы

[1] T. Schlick, *Molecular Modeling and Simulation.* Springer, first ed., 2002.

[2] R. B. Best, "Atomistic molecular simulations of protein folding," *Cur. Opin. Struct. Biol.*, vol. 22, no. 1, pp. 52 – 61, 2012.

[3] W. Humphrey, A. Dalke, and K. Schulten, "VMD: Visual molecular dynamics," *J. Molec. Graphics*, vol. 14, no. 1, pp. 33–38, 1996.

[4] C. Levinthal, "How to fold graciously," in *Mossbauer Spectroscopy in Biological Systems: Proceedings of a meeting held at Allerton House, Monticello, Illinois* (J. T. P. Debrunnder and E. Munck, eds.), pp. 22–24, University of Illinois Press, 1969.

[5] T. Creighton, *Protein Folding.* W. H. Freeman, 1992.

[6] K. A. Dill and J. L. MacCallum, "The protein-folding problem, 50 years on," *Science*, vol. 338, no. 6110, pp. 1042–1046, 2012.

[7] R. Dahm, "Friedrich Miescher and the discovery of DNA," *Dev. Biol.*, vol. 278, no. 2, pp. 274–288, 2005.

[8] O. T. Avery, C. M. MacLeod, and M. McCarty, "Studues on the chemical nature of the substance inducing transformation of pneumococcal types: induction of transformation by a desoxyribonucleic acid fraction isolated from pneumococcus type III," *J. Exp. Med.*, vol. 79, no. 2, pp. 137–158, 1944.

[9] J. Watson and F. Crick, "Molecular structure of nucleic acids," *Nature*, vol. 4356, pp. 737–738, 1953.

[10] R. E. Franklin and R. G. Gosling, "Molecular configuration in sodium thymonucleate," *Nature*, vol. 171, no. 4356, pp. 740–741, 1953.

[11] A. D. MacKerell, Jr., D. Bashford, M. Bellott, R. L. Dunbrack, Jr., J. D. Evanseck, M. J. Field, S. Fischer, J. Gao, H. Guo, S. Ha, D. Joseph-McCarthy, L. Kuchnir, K. Kuczera, F. T. K. Lau, C. Mattos, S. Michnick, T. Ngo, D. T. Nguyen, B. Prodhom, W. E. Reiher, III, B. Roux, M. Schlenkrich, J. C. Smith, R. Stote, J. Straub, M. Watanabe, J. Wiórkiewicz-Kuczera, D. Yin,

and M. Karplus, "All-atom empirical potential for molecular modeling and dynamics studies of proteins," *J. Phys. Chem. B*, vol. 102, no. 18, pp. 3586–3616, 1998.

[12] A. D. MacKerell, "Empirical force fields for biological macromolecules: Overview and issues," *J. Comput. Chem.*, vol. 25, no. 13, pp. 1584–1604, 2004.

[13] B. R. Brooks, R. E. Bruccoleri, B. D. Olafson, D. J. States, S. Swaminathan, and M. Karplus, "CHARMM: A programm for macromolecular energy, minimization, and dynamics calculations," *J. Comput. Chem.*, vol. 4, no. 2, pp. 187–217, 1983.

[14] B. R. Brooks, C. L. Brooks, III, A. D. Mackerell, L. Nilsson, R. J. Petrella, B. Roux, Y. Won, G. Archontis, C. Bartels, S. Boresch, A. Caflisch, L. Caves, Q. Cui, A. R. Dinner, M. Feig, S. Fischer, J. Gao, M. Hodoscek, W. Im, K. Kuczera, T. Lazaridis, J. Ma, V. Ovchinnikov, E. Paci, R. W. Pastor, C. Post, J. Z. Pu, M. Schaefer, B. Tidor, R. M. Venable, H. L. Woodcock, X. Wu, W. Yang, D. M. York, and M. Karplus, "CHARMM: The biomolecular simulation program," *J. Comput. Chem.*, vol. 30, no. 10, pp. 1545–1614, 2009.

[15] J. C. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R. D. Skeel, L. Kalé, and K. Schulten, "Scalable molecular dynamics with NAMD," *J. Comput. Chem.*, vol. 26, pp. 1781–1802, 2005.

[16] B. Hess, C. Kutzner, D. van der Spoel, and E. Lindahl, "GROMACS 4: Algorithms for highly efficient, load-balanced, and scalable molecular simulation," *J. Chem. Theory Comput.*, vol. 4, pp. 435–447, 2008.

[17] D. A. Case, T. E. Cheatham, T. Darden, H. Gohlke, R. Luo, K. M. Merz, A. Onufriev, C. Simmerling, B. Wang, and R. J. Woods, "The Amber biomolecular simulation programs," *J. Comput. Chem.*, vol. 26, no. 16, pp. 1668–1688, 2005.

[18] D. van der Spoel, E. Lindahl, B. Hess, C. Kutzner, A. R. van Buuren, E. Apol, P. J. Meulenhoff, D. P. Tieleman, A. L. T. M. Sijbers, K. A. Feenstra, R. van Drunen, and H. J. C. Berendsen, *GROMACS User Manual*. The GROMACS development team, 4.0 ed., 2009.

[19] A. D. MacKerell, M. Feig, and C. L. Brooks, III, "Improved treatment of the protein backbone in empirical force fields," *JACS*, vol. 126, no. 3, pp. 698–699, 2004.

[20] B. Roux and T. Simonson, *Implicit Solvent Models for Biomolecular Simulations.* Biophysical chemistry, New York, 1999.

[21] P. Ferrara, J. Apostolakis, and A. Caflisch, "Evaluation of a fast implicit solvent model for molecular dynamics simulations," *Proteins*, vol. 46, no. 1, pp. 24–33, 2002.

[22] W. C. Still, A. Tempczyk, R. C. Hawley, and T. Hendrickson, "Semianalytical treatment of solvation for molecular mechanics and dynamics," *JACS*, vol. 112, no. 16, pp. 6127–6129, 1990.

[23] T. Darden, D. York, and L. Pedersen, "Particle mesh Ewald: An N·log(N) method for Ewald sums in large systems," *J. Chem. Phys.*, vol. 98, no. 12, pp. 10089–10092, 1993.

[24] U. Essmann, L. Perera, M. L. Berkowitz, T. Darden, H. Lee, and L. G. Pedersen, "A smooth particle mesh Ewald method," *J. Chem. Phys.*, vol. 98, no. 12, pp. 10089–10092, 1993.

[25] R. D. Skeel, I. Tezcan, and D. J. Hardy, "Multiple grid methods for classical molecular dynamics," *J. Comput. Chem.*, vol. 23, pp. 673–684, 2002.

[26] R. B. Best, N.-V. Buchete, and G. Hummer, "Are current molecular dynamics force fields too helical?," *Biophys J.*, vol. 95, no. 1, pp. L07–L09, 2008.

[27] T. A. Halgren and W. Damm, "Polarizable force fields," *Cur. Opin. Struct. Biol.*, vol. 11, no. 2, pp. 236 – 242, 2001.

[28] V. Tozzini, "Coarse-grained models for proteins," *Curr. Opin. Struct. Biol.*, vol. 15, no. 2, pp. 144 – 150, 2005.

[29] C. Clementi, "Coarse-grained models of protein folding: toy models or predictive tools?," *Curr. Opin. Struct. Biol.*, vol. 18, no. 1, pp. 10–15, 2008.

[30] J. C. Kendrew, G. Bodo, H. M. Dintzis, R. G. Parrish, H. Wyckoff, and D. C. Phillips, "A three-dimensional model of the myoglobin molecule obtained by X-Ray analysis," *Nature*, vol. 181, pp. 662–666, 1958.

[31] D. Frishman and P. Argos, "Knowledge-based protein secondary structure assignment," *Proteins*, vol. 23, no. 4, pp. 566–579, 1995.

[32] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in C.* The Art of Scientific Computing, Cambridge University Press, second ed., 1992.

[33] А. В. Боресков and А. А. Харламов, *Основы работы с технологией CUDA.* ДМК Пресс, 2012.

[34] А. Боресков, Н. Марковский, and А. Харламов, *Параллельные вычисления на GPU. Архитектура и программная модель CUDA.* Издательство МГУ, 2012.