Пакет моделирования молекулярной динамики

«PUMA-CUDA»

Полное описание

Содержание

Оглавление

Общие сведения

Программа моделирования молекулярной динамики «PUMA-CUDA» (далее PUMA-CUDA) предназначена для работы в следующих вычислительных средах:

- на одном процессоре;
- на многих процессорах по технологии OpenMP;
- на одном либо двух графических процессорах согласно методу сканирования по пространству
- на одном либо двух графических ускорителях согласно составлению списка Верле, который составляется согласно методу сканирования по пространству;
- в гетерогенной вычислительной среде архитектуры N CPU * 2 GPU.

На вход программе подаётся файл параметров param.nam, структурный файл с топологией системы и силовым полем, а также может быть один или несколько .lst-файлов, которые содержат информацию о последнем состоянии системы.

1. Работа с программным комплексом

1.1 Входные данные и запуск

Для запуска программы обязательно должен присутствовать файл с параметрами. По умолчанию параметры берутся из файла param.nam. Также программа может считывать параметры из любого файла, задаваемого в качестве первого параметра при запуске, расширение не играет роли. Например:

```
Puma.exe param001.nam
```

Возможность реализована как для Windows, так и для Linux.

После программа считывает структурный файл, который задаётся параметром StrName файла параметров. Имя по умолчанию не предусмотрено.

Далее, если запуск подразумевает продолжение (значение 0 или 1 параметра Istart), то считывается файл с последним состоянием системы вида 100.lst. Вместо 100 может быть любое число, которое вычисляется по формуле NVarinat + rank, где rank – номер задачи, начиная с нуля, при запуске в режиме MPI. Значение NVariant берется из файла параметров и по умолчанию равно 100.

Можно вручную организовывать параллельные независимые запуски программы, изменяя значение NVariant.

Если Istart=0, то отсчет времени ведётся с нуля, а также создаются заново файлы траекторий и энергетических траекторий. Если Istart=1, то программа продолжает отсчет времени и продолжает записывать выходные файлы.

1.2 Основной режим работы

Основной режим работы программы сводится к непрерывному интегрированию уравнений движения с шагом, заданным параметром Tau, задаваемый в пикосекундах. Типичный шаг интегрирования 0,001 пс.

На каждом шаге программа вычисляет:

1) валентные связи;
2) валентные углы;
3) торсионные и неправильные торсионные углы;
4) кулоновское взаимодействие;
5) ван-дер-Ваальсово взаимодействие;
6) воздействие виртуальной столкновительной среды на систему (столкновительный термостат);
7) дополнительные силовые воздействия.

Под любыми взаимодействиями подразумевается расчет сил и энергии. Расчет сил необходим для интегрирования уравнений движения. Расчет энергии ведётся для справочных целей. Энергии записываются с шагом WWOut в файл энергетических траекторий.

1.3 Сведения о моделируемой системе. Структурный файл

Структурный файл – необходимое условие для запуска программы. В нём прописаны параметры силового поля, применительно к текущей системе.

Файл разделен на секции. Каждая секция начинается с заголовка секции, который, в свою очередь, начинается с символа амперсанда «&». На следующей строчке после названия секции идёт количество строк в секции.

Структурный файл содержит следующие секции:

1) &ATOMS – перечень типов атомов, включающий:
    a. название типа атома. Схоже с названием химического элемента, но это не одно и то же. Один химический элемент иметь несколько подтипов атомов, в зависимости от того, в каких связях они участвуют. Например, углерод имеет подтипы C, CA, CT.
    b. Параметры ван-дер-Ваальсового взаимодействия Rmin и Eps, используются в программе.
    c. Параметры ван-дер-Ваальсового взаимодействия Ron и Roff, не используются в программе. Вместо них используется постоянное задание размеров списков взаимодействующих атомов Rqq и RVerle.
2) &FISPEC – описание силового поля для каждого атома. Раздел включает:
    a. at – порядковый номер атома;
    b. res – номер аминокислотного остатка;
    c. t_at – ссылка на номер типа атома из таблицы ATOMS (в скобках приведено его название);
    d. t_res – ссылка номер аминокислотного остатка, в программе не используется (в скобках приведено его название);
    e. ati – номер атома в пределах остатка, в программе не используется;
    f. m_at – масса атома в а.е.м.;
    g. Q – заряд атома в Кулонах;
    h. X, Y, Z – координаты атома в начальный момент времени.
    i. f –
    j. s – уточнить у Николая Кирилловича.
3) &FSQNC – что это такое?
4) &FISV12 – список валентных связей. Строчка таблицы соответствует паре взаимодействующих атомов. Колонки по порядку означают:
    a. порядковый номер первого атома из пары;
    b. порядковый номер второго атома;
    c. порядковый номер типа связи;

d. название первого атома, служит для справочных целей, в программе не используется;

e. название второго атома, служит для справочных целей, в программе не используется.

5) &FISV13 – список валентных углов. Строчка таблицы соответствует тройке атомов, образующих углов, второй атом – цент угла. Колонки по порядку означают:

a. порядковый номер первого атома из тройки;

b. порядковый номер второго атома;

c. порядковый номер третьего атома;

d. порядковый номер типа угла;

e. разделитель «||»;

f. названия атомов, служит для справочных целей, в программе не используется.

6) &FISV14 – список торсионных углов. Строчка таблицы соответствует четверке атомов, которые образуют угол. Колонки по порядку означают:

a. порядковый номер первого атома из четверки;

b. порядковый номер второго атома;

c. порядковый номер третьего атома;

d. порядковый номер четвертого атома;

e. порядковый номер типа угла;

f. разделитель «||»;

g. названия атомов, служит для справочных целей, в программе не используется.

7) &FISVW2 – список неправильных торсионных углов. Строчка таблицы соответствует четверке атомов, которые образуют угол. Второй атом – центр неправильного торсионного угла. Колонки по порядку означают:

a. порядковый номер первого атома из четверки;

b. порядковый номер второго атома;

c. порядковый номер третьего атома;

d. порядковый номер четвертого атома;

e. порядковый номер типа угла;

f. разделитель «||»;

g. названия атомов, служит для справочных целей, в программе не используется.

8) &FPRMVW – список ван-дер-Ваальсовых взаимодействий. Каждая строчка таблицы соответствует атому определенного типа. Колонки по порядку означают:
    a. название элемента;
    b. константа epsilon;
    c. константа R_{min};
    d. текстовое описание. Как правило, содержит ссылку на литературный источник. В программе не используется;
    e. номер из исходной таблицы параметров силового поля. В программе не используется.
9) &FPRM12 – параметры валентных связей, на которые ссылается третье поле из таблицы &FISV12. Параметры служат для вычисления потенциала $U=k*(L-L_0)^2$. Список колонок:
    a. константа пружины k;
    b. начальная длина связи L_0;
    c. названия атомов, образующий валентную связь. В программе не используются.
    d. порядковый номер из силового поля программы predmd. В программе не используется.
10) &FPRM13 – параметры валентных углов, на которые ссылается четвертое поле из таблицы &FISV13. Параметры служат для вычисления потенциала $U=k*(T-T_0)^2$. Список колонок:
    a. константа жесткости k;
    b. начальный угол T_0;
    c. названия атомов, образующий валентный угол. В программе не используются.
    d. порядковый номер из силового поля программы predmd. В программе не используется.
11) &FPRM14 – параметры торсионных углов, на которые ссылается четвертое поле из таблицы &FISV14. Параметры служат для вычисления потенциала U=k*[1+Znak*cos(n*Fi), где Znak=cos(Delta). Список колонок:
    a. константа жесткости k;
    b. константа знака (фаза угла) Delta;
    c. константа n;
    d. названия атомов, образующий торсионный угол. Наименование X соответствует любому атому. В программе не используются.

e. порядковый номер из силового поля программы predmd. В программе не используется.

12) &FPRMW2– параметры неправильных торсионных углов, на которые ссылается четвертое поле из таблицы &FISVW2. Параметры служат для вычисления потенциала U=k*[1+Znak*cos(n*Fi), где Znak=cos(Delta). Список колонок:

a. константа жесткости k;

b. константа знака (фаза угла) Delta;

c. константа n;

d. названия атомов, образующий торсионный угол. Наименование X соответствует любому атому. В программе не используются. Второй атом – центральный атом неправильного торсионного улга.

e. порядковый номер из силового поля программы predmd. В программе не используется.

1.4 Файл параметров и режимы работы

Любой запуск программы начинается с чтения файла параметров. Файл параметров представляет собой текстовый файл следующей структуры.

Файл разбит на секции. Имя секции указывается в квадратных скобках. Например:

```
[main]
```

Каждый параметр внутри секции указывается на новой строчке в следующем виде:

```
Имя_параметра=значение; комментарий
```

Значение параметра может быть как числовым, так и символьным. Кавычки при использовании символьных значений не применяются.

Параметр может встретиться в каждой секции произвольное количество раз. И каждый раз ему будет присвоено значение. Таким образом параметр будет иметь значение, присвоенное ему в последний раз. Исключение – секция Holding. В этой секции каждое вновь встречающееся имя параметра будет задавать очередной элемент соответствующего массива.

В тексте приведены значения параметров по умолчанию, а также примеры.

1.4.1 Интерактивное изменение файла параметров

При работе в операционной системе Windows при изменении файла параметров программа перечитывает новые значения и сразу их применяет. Например, не перезапуская программу, можно изменить температуру или шаг численного интегрирования и сразу увидеть, как это повлияет на систему. При работе на вычислительном кластере подобный режим не уместен ввиду повышенной нагрузки на, как правило, сетевую файловую систему. Как минимум, ввиду латентности это снизило бы производительность программы. Среди других побочных действий, это замедлило бы работу других пользователей кластера.

Удобно запустить программу в одном окне (консоли), а файл параметров в текстовом редакторе в другом окне. Можно изменять параметры в текстовом редакторе, а при нажатии кнопки сохранить сразу же видеть отклик системы.

1.4.2 Главная секция

```
[main]
NVariant=100;
```

Номер единственной или первой траектории при моделировании на кластере.

```
CPUperSystem=1;      CPUs per one system in MPI
```

Количество процессоров на одну траекторию при работе на кластере. Используется в том числе и для CUDA-режима без периодических граничных условий.

```
Istart=0;            Start mode
```

Режим запуска: -1 – начальный запуск с чтением координат из структурного файла; 0 – запуск в режиме продолжения с обнулением счетчика времени, с обнулением траекторных файлов и файлов статистики, с чтением координат и скоростей из .lst-файла; 1 – запуск в режиме продолжения, с продолжением времени, с продолжением записи в траекторные файлы и файлы статистики.

```
VWCoulombCombined=1;1 - together on 1 GPGPU accelerator
```

Выполнять расчет ван-дер-Ваальсовых и кулоновских взаимодействий на одном ускорителе при наличии нескольких устройств. Реализовано для работы с периодическими граничными условиями. Эффективно при запуске нескольких траекторий на узле с

несколькими графическими картами. При этом производительность на ускоритель ниже, но суммарная производительность на узел повышается.

```
StrName=dna2.str;   Name of the str-file
```

Имя структурного файла. Обязательный параметр.

```
Tau=0.001;                Main integrity constant
```

Шаг численного интегрирования в пс.

```
CUDA_N=1;
```

Выбор текущего графического ускорителя. Не всегда работает по причине драйверов.

```
;       ============ Thermostat
TStart=001;               Start temperature
```

Начальная температура в К. Разыгрываются начальные скорости атомов при режиме запуска IStart=-1.

```
TRef=280;           Termostat temperature
```

Заданная температура системы в К.

```
dTRank=20;            TRef+=dTRank*rank;
```

Инкремент температуры в К при запуске расчета нескольких траекторий на вычислительном кластере. При этом заданная температура будет выглядеть как TRef+=dTRank*rank, где rank – номер процесса в режиме запуска MPI-задачи. Первая задача имеет номер rank=0, поэтому её температура термостата будет соответствовать TRef.

```
dTRef=0;
```

Инкремент температуры термостата в К за 1 пс. Используется в опытах по линейному нагреванию.

```
Lamd=10;
```

Параметр термостата лямбда. Частота столкновений с виртуальными частицами. Масса виртуальной частицы принимается равно 1 а.е.м.

```
EpsilonQr=1;
```

Параметр кулоновского взаимодействия. Обратное значение диэлектрической проницаемости среды. Энергия и сила кулона умножается на это значение. Если EpsilonQr=0, сила кулона будет равна нулю. Именно из-за этого выбрано обратное значение параметра.

```
MoveToCenterMass=0;
```

Опция включена, если значение параметра равно 1. Перемещает центр масс на каждом шаге численного интегрирования уравнений движения к предыдущему его значению.

1.4.3 Выходные данные

В качестве выходных файлов во всех режимах используются: траекторные файлы вида 100g.trj; файлы последних значений координат, скоростей, размера расчетной ячейки и др. параметров вида 100.lst; файлы статистики вида 100ww.txt; .pdb-файлы вида 10000000.pdb. Здесь везде 100 означает номер системы, начиная с параметра NVariant, а остальные значащие цифры в имени .pdb-файлы означают время в пикосекундах.

```
;       ============ Output
OutTrj=water.trj;         Trajectory
```

Имя файла траектории. Не используется

```
OutLst=pgb.lst;           LastFile
```

Имя .lst-файла. Не используется

```
Nterm=100;                Output to terminal
```

Период времени вывода на экран в шагах.

```
Nlst=1000;                Save LastFile
```

Период записи в .lst-файл в шагах.

```
Ntrj=1000;                Output to trajectory file
```

Период записи в траекторный файл в шагах.

```
WriteTrjExt=0;      Output extended components to trajectory
```

Флаг вывода в траекторию расширенных компонент. Если WriteTrjExt=1, то расширенные компоненты выводятся. По умолчанию WriteTrjExt=0.

```
AllSteps=15000000;        Steps in one run
```

Количество шагов за один запуск после которого программа завершит работу.

```
PDBOut=1000;
```

Период записи в .pdb-файл в шагах. Не может быть меньше 1000 ввиду ограничений на имя файла.

```
WWOut=100;
```

Период записи в файл статистики в шагах.

```
MaxTimeH=11;
MaxTimeM=59;
MaxTimeS=00;
```

Максимальное время счета в часах, минутах и секундах. По окончании этого времени выполнение программы завершается. Создаётся last-файл. На экран выводится сообщение Time is out. Чтобы не пользоваться остановкой по истечению какого-то времени, необходимо занулить все параметры MaxTimeH, MaxTimeM, MaxTimeS. О молчанию они равны нулю, т.е. опция остановки программы по таймеру не используется.

1.4.4 Параметры невалентных взаимодействий

```
;       ============ NonValent
Rqq=10.5;                 R Coulomb cut-off
```

Радиус кулоновских взаимодействий в Å.

```
Rvw=10.5;           R van der Waals cut-off
```

Радиус ван-дер-Ваальсовых взаимодействий в Å.

```
RVerle=12.5;          R Verle List cut-off
```

Критерий пересчета списка Верле в Å. Рекомендуется брать на 2 Å больше максимального радиуса взаимодействия.

```
RVerleDelta=0.1;
```

Приращение радиуса пересчета списка Верле в Å. Используется для алгоритма оптимизации размера составления списка Верле. Сложно предположить другое значение. Поэтому можно не указывать в файле параметров.

1.4.5 Управляемая молекулярная динамика (силовые воздействия)

1.4.5.1 Воздействие постоянной силой

```
;       ============ FORCES
;       ============ Constant force
[ForceConstant]
Enabled=0;            Constant Force 1-on/0-off, 1-X, 2-Line
```

Режим постоянной силы. 0 – выключена, 1 – приложена вдоль оси X, 2 – прриложена вдоль оси, соединяющей два атома.

```
Value=-300;            Value of the Force
```

Значение силы в пН.

```
AtomLeft=449;           Number of force applied atoms
AtomRight=1442;
```

Номера атомов, к которым применяется сила. Если левый атом левее правого, значение силы указывается отрицательное.

1.4.5.2 Растяжение с постоянной скоростью

В начале применения силы в точках приложения силы ставятся два кронштейна (стенки), которые разводятся с постоянной скоростью. Атомы системы AtomLeft и AtomRight (из секции ForceConstant) соединены с кронштейнами пружинами жесткостью K и нулевой начальной длиной.

```
;       ============ Constant speed
[ForceV]
Enabled=1;            Constant Force 1 - by X, 2 - affinne, 3
- by the line, 4 - affine by the line
```

Режим растягивания: 0 – выключен, 1 – вдоль оси X, 2 – вдоль оси X с аффинным растяжением системы, 3 – вдоль оси, соединяющей два атома, 4 – вдоль оси, соединяющей два атома, с аффинным растяжением системы.

Аффинное растяжение системы (ровно как и пространства) введено с целью более быстрой реакции системы на растяжение, которое приложено к концам.

```
Value=0.1;            Value of the Force
```

Значение силы в А/пс.

```
K=100;           Number of force applied atoms
```

Константа жесткости пружины.

```
Cycled=0;      1=Enabled cylce force with period:
```

Режим циклической силы: 0 – выключен, 1 – включен.

```
Period=1000;
```

Полный период циклической силы в пс.

1.4.6 Сфероцилиндр

Имеется возможность ограничить систему сфероцилиндром с отталкивающими стенками, чтобы испаряющиеся молекулы и атомы не разлетались на бесконечность.

Не рекомендуется использовать совместно с периодическими граничными условиями. Однако, технически это позволено.

```
;       ============ Cylinder
[Cylinder]
Enabled=0;
```

Режим работы: 0 – выключен, 1 – включен.

```
H=150;
```

Длина сфероцилиндра в Å.

```
R=50;
```

Радиус сфероцилиндра в Å.

```
Rmin=1;
```

Параметр R_{min} сфероцилиндра в Å.

1.4.7 Периодические граничные условия

```
;       ============ Periodic boundary conditions
[PBC]
Enabled=1;
```

Режим работы с периодическими граничными условиями: 0 – выключен, 1 – включен.

```
AX=127.479;
AY=38.24;
AZ=121.105;
```

Размеры расчетной ячейки в Å.

1.4.8 Баростат. NPT-ансамбль

Параметры баростата задаются в секции PBC (периодических граничных условий).

```
BarostatEnabled=0;
```

Режим работы баростата: 0 – выключен, 1 – включен.

```
StatisticSize=5;
```

Ширина окна усреднения давления.

```
bettaPx=0;
bettaPy=0;
bettaPz=0;
```

Приращение давления по соответствующей компоненте.

```
Refx=0.0001;
Refy=0.0001;
Refz=0.0001;
```

Заданное давление по соответствующим осям в ГПа. 0,0001 ГПа = 100 КПа = 1 атм.

1.4.9 Приведение системы к заданной плотности

```
;       ============ Ro
[Ro]
Enabled=0;
```

Включения режима приведения системы к заданной плотности: 0 – выключен, 1 – включен.

```
RoDestination=1960;
```

Заданная плотность в кг/м^3.

```
RoAccuracy=0.5;
```

Погрешность в кг/м^3.

```
Dx=0.00005;
Dy=-0.00000;
Dz=-0.00000;
```

Относительное приращение соответствующих размеров расчетной ячейки на каждом шаге. Приращение остановится, если плотность системы достигнет заданной, либо значений из следующих параметров:

```
DestinationX=0;
DestinationY=0;
DestinationZ=0;
```

Заданные значения в Å размеров расчетной ячейки.

1.4.10 Режим расчета взаимодействия двух подсистем

```
;       ============ Two Systems Interaction
[TwoSystemsInteraction]
Enabled=0;
```

Включение режима: 0 – выключен, 1 – включен.

При включении данного режима в файл статистики будут выводиться отдельные значения энергии для первой и второй подсистем

```
FirstAtomLeft=1;
```

Первый атом первой подсистемы.

```
FirstAtomRight=960;
```

Последний атом первой подсистемы. Включительно.

```
SecondAtomLeft=961;
```

Первый атом второй подсистемы.

```
SecondAtomRight=1921;
```

Последний атом второй подсистемы. Включительно.

```
WriteToTrjOnlyFirstSystem=1;
```

Выводить в траекторный файл только первую подсистему. Если опция равна 1, то в траекторный файл пишутся атомы с FirstAtomLeft по FirstAtomRight включительно.

```
ForceVToSecondSystemType=0;         Constant speed
stretching: 1 by the X-axis, 2 - by the line, 3 - by Direction,
4-Force by line
```

Режим силового воздействия на вторую подсистему. При включении этого режима силовое воздействие удерживает первую подсистему за центр масс, а к центру масс второй подсистемы прилагает силовое воздействие, обеспечивающее постоянную скоростью

последней за счет гуковской пружины, аналогичной режиму растяжения с постоянной скоростью.

0 – режим выключен, 1 – растяжение вдоль оси X, 2 – растяжение вдоль прямой, соединяющей центры систем, 3 – растяжение вдоль направления, указанного параметрами Direction, 4 – воздействие силы, приложенной к центру масс.

```
Speed=0.1;                              Force speed (A/ps)
```

Скорость растяжения в Å/пс

```
K=100;                                  Hook coefficient
```

Жесткость пружины.

F=0;

Значение постоянной силы, которая прикладывается к центру масс первой и второй подсистем с противоположным знаком.

```
DirectionX=1.0;
DirectionY=0.0;
DirectionZ=0.0;
```

Направление растяжения. Имеет смысл для режима 3 – растяжение вдоль направления, указанного параметрами Direction.

1.4.11 Силовые воздействия, направленные на группы атомов. МД-конструктор

Каждое силовое воздействие задается группе атомов.

Возможны три типа силовых воздействий:

- Реализация формулы 1. Атомы будут лежать в указанной плоскости.
- Жесткая фиксация атомов. Обнуление их скоростей.
- Расположение пар атомов на определенном расстоянии друг от друга.

```
;       ============ Holding
; Potential for holding atoms
; U = 0.5*En*(ri*e - r0 * e)**2; F = -En * (ri*e - r0 *
e)*e; r, r0, e, F - vectors. (Формула 1)
[Holding]
N=2; 242 - forces, others - without forces
```

Количество групп атомов. Если параметр равен нулю, то силовые воздействия выключены

```
En=10.0;
```

Параметр En в формуле 1.

```
Type=0;
```

Тип поведения. 0 – реализация формулы 1; 1 – фиксация атома; 2 – расположение пар атомов на определенном расстоянии друг от друга.

```
I1=1;
```

Начальный атом

```
IPeriod=320;
```

Период

```
ISize=3;
```

Количество атомов, либо пар атомов

Таким образом, будут выбраны атомы 1, 1+320, 1+2*320.

```
R0=-25.189 0.0 0.0;
```

Параметр-вектор формулы 1. Точка, через которую проходит плоскость. Задается в виде трёх чисел,

```
E=1.0 0.0 0.0;
```

Вектор-нормаль к плоскости.

При использовании Type=2 осуществляется стягивание пар атомов на расстоянии I2L. Стягиваются ISize пар атомов (I1, I2), (I1+ IPeriod, I2+ I2Period), …

```
I1=32;
```

Начальный номер первого атома в паре.

```
IPeriod=32;
```

Приращение номера первого атома.

```
I2=1912;
```

Начальный номер второго атомов в паре.

```
I2Period=-32;
```

Приращение номера второго атома.

```
I2L=2.0;
```

Поддерживаемое расстояние между атомами.

```
ISize=30;
```

Количество пар атомов

```
;R0=-15.0 0.0 0.0;
```

При Type=2 не используется.

```
;E=1.0 0.0 0.0;
```

При Type=2 не используется.

1.5 Компиляция программы PUMA-CUDA.

Компиляция программы проходит в два этапа. Сначала компилируются функции, реализованные на графическом ускорителе компилятором nvcc. Данный этап может отсутствовать (см. директивы условной компиляции). Затем компилируется программа компилятором языка C++ с поддержкой MPI, либо без него.

```
nvcc -arch=compute_70 -c kernel.cu -O3
mpiCC *.cpp kernel.o -O3 -L/common/cuda-10.0/lib64 -
I/common/cuda-10.0/include -lcudart -std=c++0x
```

Здесь вместо compute_70 может быть указана используемая архитектура.

Путь к библиотекам показан примерный.

1.5.1 Директивы условной компиляции

Программист/системный администратор может управлять директивами условной компиляции. Они настраиваются путём изменения нижеперечисленных констант в файле mdsystem.h.

```
//#define COMPILE_FPGA
```

Компиляции программы с использованием ПЛИС. По умолчанию возможность отключена (закомментирована).

```
#define COMPILE_CUDA
```

Компиляции программы для работы на графических ускорителях. Комментируется для использования на компьютерах без GPU.

```
//#define MYSQL
```

Поддержка записи в СУБД MYSQL. Работает в пробном режиме для доказательства возможности записи программы, которая выполняется на вычислительном кластере, напрямую в СУБД на сервере в Интернете.

```
//#define MPI_CPU
```

Выполнение одной задачи несколькими процессами. Выигрыш ниже линейного ввиду латентности сети.

```
#define USE_CUDA   1
```

Использовать графический ускоритель.

Директивы COMPILE_CUDA и USE_CUDA отвечают за разные возможности. Первая включает файл, который обязан компилироваться компилятором NVIDIA, а вторая разрешает считать на ускорителе: 0 – считать на CPU, 1 – на GPU.

```
#define USE_VERLET 1
```

Использовать алгоритм Верле построения списков взаимодействующих пар атомов. На графических ускорителях даёт наилучшие результаты. Работает и на графических процессорах, а на центральных. Несмотря на то, что используется также алгоритм сканирования по пространству, константу USE_LIST необходимо выключать (значение 0).

```
#define USE_LIST   0
```

Использовать только алгоритм сканирования по пространству.

```
#define USE_THREADS 1
```

Использовать многопроцессорные технологии для одновременного вычисления невалентных взаимодействий на графическом процессоре и валентных взаимодействий на

ядрах центрального процессора. Использует стандарт языка C++11. Включается параметром компиляции -std=c++0x.

Для расчета ван-дер-Ваальсовых и кулоновских взаимодействий на двух разных ускорителях в файле cuh.h применяется константа

```
#define SECOND_DEVICE 0
```

0 – использовать один ускоритель, 1 – использовать два ускорителя.

```
#define WIN321
```

Использовать ли сетевые возможности для работы в связке PUMA-CUDA – TAMD. Также включает интерактивную работы с файлом параметров. Работает только в Windows (название происходит от Win32). Возможность реализована для интерактивной молекулярной динамики: на запущенной программе пользователь меняет файл параметров, программа PUMA-CUDA сразу же считывает параметры, а программа TAMD выводит координаты моделируемой системы в режиме реального времени.

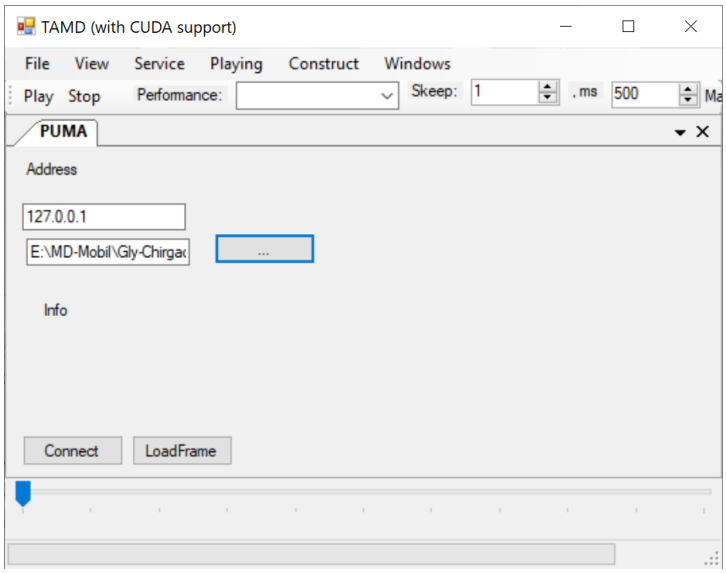
1.6 Интерактивный запуск связки PUMA-CUDA – TAMD

Программа моделирования молекулярной динамики PUMA-CUDA способна работать в связке с Анализатором траекторий молекулярной динамики TAMD. Данный режим работы реализован слежением за системой в режиме реального времени.

Ещё это можно назвать как клиент-серверный режим работы по протоколу TCP/IP. В качестве сервера выступает программа моделирования молекулярной динамики. Клиентов может быть несколько (текущее ограничение до 10, но это ограничение легко расширить). Для работы с каждым клиентом создается отдельный вычислительный поток (или поток операционной системы). Это сделано для того, чтобы производительность процесса расчетов не страдала во время медленной сетевой передачи. Для этой цели желательно иметь лишнее свободное ядро центрального процессора. Но совсем не обязательно иметь столько ядер, сколько будет клиентов, достаточно только одного на всех клиентов. Процесс передачи координат не ресурсоемкий.

Изначально запускается программа PUMA-CUDA. Компиляция должна производиться с включенной директивой WIN321. Возможность работает только в ОС Windows. Хотя принципиальных трудностей реализации для ОС семейства Linux нет. Уже сейчас программа моделирования и программа-анализатор могут находиться на разных компьютерах, соединённых по сети.

При запуске важно имя или сетевой адрес компьютера, на котором запущена программа PUMA-CUDA. Если для расчетов и для анализа используется один и тот же компьютер, то достаточно использовать имя localhost (или адрес 127.0.0.1).



Далее запускается TAMD. В Анализаторе необходимо выбрать меню File, пункт PUMA-connect… Заполняются поля адрес (по умолчанию 127.0.0.1, т.е. локальный компьютер) и имя pdb-файла. PUMA-CUDA поставляет только координаты системы для экономии сетевого траффика. После нажатия кнопки Connect осуществляется попытка сетевого соединения с программой PUMA-CUDA. Для получения кадра существует кнопка Load Frame. Предварительно лучше открыт режим трехмерного просмотра, меню View – 3D Vision (или использовать сочетание клавиш Control-F3).

Для постоянной передачи координат и визуализации системы в режиме реального времени рекомендуется задать паузу между кадрами в районе нескольких сотен миллисекунд (параметр после слова «ms») и нажать кнопку Play. Значение 500 мс

соответствует двум кадрам в секунду. Процесс моделирования не столь быстрый. Поэтому нескольких или одного кадра и даже меньше кадров в секунду будет вполне достаточно для отслеживания состояния системы.

Для остановки передачи координат служит кнопка Stop. Она останавливает визуализацию. Моделирование при этом продолжается.

Установка слишком малого значения паузы между кадрами (например, 1 мс) может привести к излишней нагрузке на вычислительную установку.

1.6.1 Реализация

Программа моделирования PUMA-CUDA выступает в качестве сервера, прослушивая порт 20001 протокола TCP/IP (задаётся в переменной int MY_PORT). Каждый клиент обслуживается в отдельном вычислительном потоке, их общее количество задаётся в директиве MaxThreads, которая в исходно версии принимает значение 10.

Передача данных происходит в режиме запрос – ответ. Запрос должен исходить по инициативе клиента (Анализатора траекторий молекулярной динамики).

И запросы, и ответы передаются в текстовом виде для простоты отладки.

На данном этапе поддерживается две команды: exit и GetFrame. Команды чувствительны к регистру.

Получив команду exit, сервер закрывает соединение с клиентом.

По команде GetFrame передаётся N+1 строка ответа, где N – количество атомов в системе. В первой строке передаётся информация о системе:

```
Size Step
```

где Size – количество атомов в системе, а Step – шаг численного интегрирования. Получив размерность, клиент знает, сколько ещё строк ожидается в ответе. Последующие строчки содержат координаты атомов:

```
x y z
```

Как видно из примера, координаты передаются через пробел. По три координаты на каждый атом.

Возможности программы легко расширять, сохраняя совместимость с предыдущими версиями. К примеру, в первой строке можно передавать любую другую информацию о системе: энергии, температуру, давление. А при передаче координат можно ещё посылать и скорость атомов.

1.7 Параметры командной строки

По умолчанию (без указания параметров командной строки) программа работает с файлом параметров param.nam.

После имени программы можно указать произвольный файл параметров. Если он существует, то программа будет работать с ним. Если его нет, то программа будет считывать параметры из файла param.nam.

1.8 Запуск на вычислительных кластерах (в MPI-среде)

При работает в на вычислительных кластерах, также называемых суперкомпьютерами, в MPI-среде программа запускает n независимых реализаций

вычислительного эксперимента. Число n задается параметром -np команды запуска mpiexec.

Если нет параметром командной строки программы PUMA-CUDA (не путать с параметрами команды mpirun), то по умолчанию считывается файл параметров param.nam.

Если параметр командной строки PUMA-CUDA один, то считывается именно указанный файл параметров для всех реализаций. Если такого файла нет, то программа будет считывать параметры из файла param.nam.

Если параметров командной строки несколько, то предполагается, что каждой реализации соответствует свой файл параметров. В этом случае параметров командной строки должно быть столько же, сколько и число реализаций n.

Первая реализация получает номер варианта, указанный в параметре NVariant файла параметров. Номера остальных реализаций получают последовательные номера. MPI-программы запускаются независимо. И каждая из них прибавляет свой номер (ранк) к параметру NVariant. Ранки отсчитываются от нуля.

Если в параметрах командной строки PUMA-CUDA указано несколько файлов параметров, то достаточно в каждом файле параметров указывать один и тот же NVariant, т.е. каждая реализация прибавляет 1 к этой переменной.

При чтении текста может возникнуть путаница, что такое параметр командной строки. Когда говорится о команде mpirun, то все её параметры, включая имя исполняемого модуля PUMA-CUDA, являются параметрами. Но когда речь идёт о самой программе PUMA-CUDA, то подразумеваются именно её параметры, которые нумеруются с единицы.

1.8.1 Работа с несколькими ускорителями

Программа может задействовать два ускорителя для одновременного расчета сил Ван-дер-Ваальса и кулона. Для включения этой возможности необходимо установить значение 1 константе SECOND_DEVICE в файле cuh.h. На каждом узле вычислительного кластера (или единственной машины) должно присутствовать минимум 2 укорителя.

К моменту написания инструкции более эффективным признан режим расчета Ван-дер-Ваальсовых и кулоновских взаимодействий в рамках одной функции ядра (kernel). Для включения этой возможности в файл параметров в секцию [main] необходимо добавить строчку VWCoulombCombined=1.

При работе на вычислительном кластере, на каждом узле которого установлено несколько ускорителей, наиболее эффективно будет запускать столько реализаций программы, сколько установлено ускорителей. Для этого необходимо указывать параметр -ppn команды mpirun. Каждая реализация будет поочередно использовать все укорители узла.

При работе с малыми системами с целью экономии вычислительных ресурсов можно устанавливать параметр -ppn команды mpirun как удвоенное количество узлов. В этом случае на каждом ускорителе будет считаться по две задачи. Режим имеет смысл использовать, когда загрузка ускорителя составляет менее 100 %.

Ограничением сверху количества задач, исполняемых одним ускорителем, является размер видеопамяти

Загрузку видеоускорителя и используемый объем видеопамяти (глобальной памяти видеоускорителя) можно проверить командой nvidia-smi:

```
~$ nvidia-smi

Fri Jan 12 12:17:00 2024

+-----------------------------------------------------------------------------+
| NVIDIA-SMI 470.199.02   Driver Version: 470.199.02   CUDA Version: 11.4     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  NVIDIA GeForce ...  On   | 00000000:07:00.0 Off |                  N/A |
| 30%   49C    P2   111W / 200W |    732MiB /  7979MiB |     88%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0   N/A  N/A    605323      C   ./pumacuda                        729MiB |
+-----------------------------------------------------------------------------+
```

Жирным текстом выделены:

- 732 МБ – объем видеопамяти, используемый операционной системой;
- 7979 МБ – общий объем видеопамяти;
- 88% – степень загрузки видеоускорителя;
- 729 МБ – объем видеопамяти, используемый программой.

Степень использования графического процессора и памяти видеоускорителя – параметры динамические. Их полезно смотреть во времени. Для этого можно использовать команду:

```
watch nvisia-smi
```

1.9 Примеры команд запуска PUMA-CUDA

Пусть исполняемый модуль будет называться pumacuda в ОС Linux и pumacuda.exe в ОС Windows.

Для запуска программы с файлом параметров по умолчанию (param.nam) в ОС Windows достаточно выполнить команду:

```
pumacuda.exe
```

а в ОС Limux:

```
./pumacuda
```

Остальные примеры будут относиться к ОС Linux. В ОС Windows они будут иметь похожий вид.

Запуск с конкретным файлом параметров param_DNA.nam:

```
./pumacuda param_DNA.nam
```

Запуск программы в фоновом режиме, чтобы программа продолжала расчет, когда пользователь отключился бы от терминала:

```
nohup ./pumacuda param_DNA.nam &
```

В этом режиме вывод на экран направляется в файл nohup.out.

Для периодического вывода на экран файла nohup.out можно использовать команду:

```
watch tail nohup.out
```

Для запуска на вычислительном кластере в MPI-среде служит команда mpirun (а также команда mpiexec, в зависимости от версии программного обеспечения).

Запуск 8 задач на 2 узлах, по 4 ускорителя на каждом с файлом параметров по умолчанию (param.nam) на 10 минут:

```
mpirun -np 8 -ppn 4 -maxtime 10 pumacuda
```

Запуск 8 задач на 2 узлах, по 4 ускорителя на каждом на 720 минут, каждая со своим файлом параметров (param_1.nam param_2.nam param_3.nam param_4.nam):

```
mpirun -np 8 -ppn 4 -maxtime 10 pumacuda param_1.nam param_2.nam param_3.nam param_4.nam
```

1.20 Выходные данные

1.20.1 pdb-файлы

Программа выводит pdb-файл всей системы каждые PDBOut шагов (параметр секции [main]). Если PDBOut=0, то PDB-файлы не выводятся.

Наименование файла соответствует номеру варианта моделируемой системы и моменту времени в пикосекундах. Он вычисляется как сумма NVariant (параметр секции [main]) и номер ранка процесса MPI (отсчёт ведётся с нуля). Таким образом, если NVariant=100, то первая реализация системы будет иметь номер 100, вторая 101 и т.д.

На момент времени выделяется 5 знаков. Таким образом, через 200 пс имя pdb-файла первой системы составит 10000200.pdb.

Каждая строка pdb-файла содержит сведения об атомах и их координатах. Последние две строки – содержат сведения о системе.

Формат вывода файла:

- Столбцы 1-4 – слово «ATOM».

- 7-11 Atom serial number right integer
- 13-16 Atom name left * character
- 17 Alternate location indicator character
- 18-20 Residue name right character
- 22 Chain identifier character
- 23-26 Residue sequence number right integer
- 27 Code for insertions of residues character
- 31-38 X orthogonal Angstrom coordinate right floating
- 39-46 Y orthogonal Angstrom coordinate right floating
- 47-54 Z orthogonal Angstrom coordinate right floating
- 55-60 Occupancy right floating
- 61-66 Temperature factor right floating
- 73-76 Segment identifier (optional) left character
- 77-78 Element symbol right character
- 79-80 Charge (optional) character

1.20.1 Траекторные файлы

1.20.1 Файлы энергетических траекторий

2. Алгоритмы и их реализация

Основное предназначение программы моделирования молекулярной динамики – расчет энергий и сил, действующих на каждый атом.

Энергии (потенциалы) носят справочный характер. Они нужны для вычисления макроскопических величин, таких как температура и суммарные компоненты энергии (энергия кулоновского взаимодействия, ван-дер-Ваальсового, валентных связей и др.).

Всех сил и потенциалов осуществляется на каждом шаге численного интегрирования.

2.1 Расчет сил и потенциалов валентных взаимодействий

2.1.1 Валентные связи

Список валентных связей в программе соответствует списку Fisv12 структурного файла. Параметры силовых взаимодействий берутся из списка Fprm12 структурного файла, на который ссылаются строчки списка Fisv12.

Каждая строка списка валентных связей даёт одно значение потенциала и две силы, действующие противоположно на каждый атом валентной пары.

Потенциал валентных связей рассчитывается по формуле:

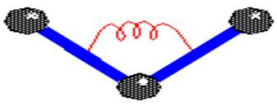
$$U_{bonds} = \sum_{i=1}^{Nbonds} k_i (r_i - r_{eq\,i})^2$$

где Nbonds – количество валентных связей, соответствующее количеству строк в списке Fisv12 структурного файла; k_i – [пН/Å] константа жесткости из справочника Fprm12; r_i – [Å] вычисляемое на каждом шаге расстояние между парой атомов; $r_{eq\,i}$ – [Å] равновесная длина пружины из справочника Fprm12.

Сложность операции линейна и не занимает много времени.

Вычисление валентных связей, как и остальных валентных взаимодействий может быть вычислено независимо от всех остальных силовых взаимодействий. Здесь и далее по тексту это важное замечание, относящееся к организации параллельного вычислительного процесса.

2.1.2 Валентные углы



Список валентных углов в программе соответствует списку Fisv13 структурного файла. Параметры силовых взаимодействий берутся из списка Fprm13 структурного файла, на который ссылаются строчки списка Fisv13.

Каждая строка списка валентных углов даёт одно значение потенциала и три силы, действующие на каждый атом валентного угла.

Потенциал валентных углов рассчитывается по формуле:

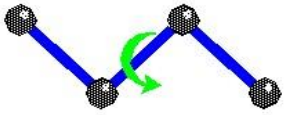
$$U_{angles} = \sum_{i=1}^{Nangles} k_{\Theta i}(\Theta_i - \Theta_{eq\,i})^2$$

где Nangles – количество валентных углов, соответствующее количеству строк в списке Fisv13 структурного файла; $k_{\Theta i}$ – [пН/градус] константа жесткости из справочника Fprm13; Θ_i – [градус] вычисляемое на каждом шаге значение угла, образованного тройкой атомов, номера которых указаны в справочнике Fprm13; $\Theta_{eq\,i}$ – [градус] равновесное значение угла из справочника Fprm12

Сложность операции линейна и не занимает много времени.

Вычисление валентных углов, как и остальных валентных взаимодействий может быть вычислено независимо от всех остальных силовых взаимодействий.

2.1.3 Торсионные углы



Список торсионных и неправильных торсионных углов в программе соответствует спискам Fisv14 и Fisvw2 структурного файла. Параметры силовых взаимодействий берутся из списков Fprm14 и Fprmw2 структурного файла, на который ссылаются строчки списков Fisv14 и Fisvw2.

Каждая строка списка торсионных углов даёт одно значение потенциала и четыре силы, действующие на каждый атом валентного угла.

Потенциал торсионных углов рассчитывается по формуле:

$$U_{dihedral} = \sum_{i=1}^{Ndihedrals} \frac{V_i}{2}(1 + \cos(n_i\phi_i)^2$$

где Ndihedrals – количество торсионных углов, соответствующее количеству строк в списке Fisv14 структурного файла; V_i [пН] и n_i– константы из справочника Fprm14; Θ_i – [градус] вычисляемое на каждом шаге значение угла, образованного четверкой атомов, номера которых указаны в справочнике Fprm14.

$$U_{dihedral} = \sum_{i=1}^{Ndihedrals} k_{\psi i}(1 - \cos(2\psi_i))^2$$

Сложность операции линейна. Занимает больше времени, чем вычисление валентных связей и валентных углов.

Вычисление торсионных углов, как и остальных валентных взаимодействий может быть вычислено независимо от всех остальных силовых взаимодействий.

2.1.4 Список валентных взаимодействий

Для расчета невалентных взаимодействий необходимо исключить из рассмотрения те пары атомов, которые участвуют в валентных взаимодействиях.

Этот список хранится в памяти ЭВМ как линеаризованный зубчатый массив. Организация зубчатых массивов обсуждается в главе, посвященной спискам Верле. Такая структура особенно хорошо подходит по нескольким причинам:

1. Каждый атом может участвовать в произвольном количестве пар. Ввиду этого обстоятельства прямоугольный массив обладал бы избыточностью.
2. Зная атом, легко можно определить всех его соседей без необходимости просмотра всего списка.
3. Линеаризованный массив намного удобнее и быстрее копировать в память графического ускорителя, в отличии, например, от связанных структур (от списков и деревьев).

Заполнение списка пар валентных взаимодействий осуществляется в начале работы программы, после загрузки структурного файла. Этот список не меняется на протяжении всего моделирования. В молекулярной динамике не меняется топология системы, первичная структура белка, не разрушаются и не создаются химические связи. Хотя, теоретически, изменить его можно.

2.2 Невалентные взаимодействия и алгоритмы работы в пространстве

2.2.1 Сила и энергия кулона

Энергия кулона рассчитывается между парами атомов. Ограничение в первой сумме написано для уменьшения вычислений в 2 раза: согласно третьему закону Ньютона, сила, действующая со стороны одного атома на другой, равна по модулю и противоположна по знаку силе, действующей со стороны второго атома на первый.

$$U_{qq} = \sum_{i<j} \sum_{j} \frac{q_i q_j}{\varepsilon \cdot r_{ij}} \cdot W_{qq}(r_{ij})$$

Сила вычисляется как производная от энергии.

2.2.2 Сила и энергия Ван-дер-Ваальса

Энергия Ван-дер-Ваальса рассчитывается между парами атомов. Ограничение i<j в первой сумме написано для уменьшения вычислений в 2 раза: согласно третьему закону Ньютона, сила, действующая со стороны одного атома на другой, равна по модулю и противоположна по знаку силе, действующей со стороны второго атома на первый.

$$U_{VdW} = \sum_{i<j} \sum_{j} 4\varepsilon_{ij} \left[\left(\frac{\sigma_{ij}}{r_{ij}} \right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}} \right)^{6} \right] \cdot W_{VdW}(r_{ij})$$

$$W_{VdW}(r_{ij}) = \begin{cases} 1, r_{ij} \le R_{on} \\ \left(R_{off}^2 - r_{off}^2\right)^2 \cdot \left(R_{off}^2 - 3R_{on}^2 + 2r_{ij}^2\right)^2, R_{on} < r_{ij} < R_{off} \\ 0, r_{ij} \ge R_{off} \end{cases}$$

Где:

- U_{VdW} –энергия Ван-дер-Ваальсовых взаимодействий;
- ε, σ – константы;
- r_{ij} – расстояние между атомами i и j;
- W_{VdW} (r_{ij}) — функция переключения, гладко сводящая потенциал в ноль на участке [R_{on}, R_{off}];
- R_{on} – координата начала затухания энергии;
- R_{off} – координата полного затухания энергии.

Сила вычисляется как производная от энергии.

!!! 2^(1/6)*singma=RMin

Добавить формулы из 2.14 стр. 9 книги МД.

Под алгоритмами работы в пространстве понимаются алгоритмы увеличения скорости расчета невалентных взаимодействий.

Если предположить, что все частицы взаимодействуют со всеми, то сложность вычисления невалентных взаимодействий оценивается как $O(N^2)$. Точнее, нужно обработать $(N-1)^2/2$ пар взаимодействий, где N – количество атомов в системе. Каждая частица не взаимодействует сама с собой, отсюда вычитание единицы. Учёт третьего акона Ньютона дарит нам деление количества вычислений пополам.

Силы, действующие на каждый атом, могут быть найдены независимо от сил, действующих на другие атомы. Таким образом, задача пригодна для решения методом параллельного программирования. Максимальное количество потоков может достигать количества атомов N.

Даже при простейшей реализации расчетов невалентных взаимодействий, когда все атомы взаимодействуют со всеми, в параллельной вычислительной среде приходится пренебрегать третьим законом Ньютона. Таким образом вычислительная сложность возрастает в два раза до $O((N-1)^2)$. В одном вычислительном потоке учитывается лишь действие окружения на рассматриваемый атом. Действие атома на окружение в конкретном вычислительном потоке не вычисляется (хотя оно уже вычислено, осталось лишь поменять

знак). Это следует из нежелания делать довольно масштабные взаимодействия между потоками, что является одной из главных причин замедления параллельных программ.

2.2.1 Ограниченные потенциалы

Будем исходить из предположения, что каждый атом взаимодействует лишь со своим окружением, которое ограничено радиусом взаимодействия (параметр Rqq в секции main фала параметров). Тогда сложность задачи вычисления невалентных взаимодействий будет порядка O(NM/2), где M – число атомов, попадающих в радиус взаимодействия. С возрастанием количества частиц число M не меняется, т.к. оно зависит только от радиуса Rqq. Таким образом мы получаем линейную сложность задачи вместо квадратичной.

Разумеется, число атомов, взаимодействующих с данным, зависит от плотности системы. Но учитывая тот факт, что увеличения числа атомов связано с желанием работать с б*о*льшими системами одинакового класса молекул, то число M можно считать приблизительно постоянным. При радиусе 12,5 Å оно составляет порядка 500 для органических систем.

2.2.2 Алгоритм сканирования по пространству

Алгоритм сканирования по пространству заключается в разделении всего пространства на ячейки.

Пусть система занимает область, ограниченную прямоугольным параллелепипедом. Сама система может и не быть прямоугольной. Речь идёт о наименьшем прямоугольном параллелепипеде, который может вместить систему. Грани параллелепипеда параллельны осям.

Этот прямоугольный параллелепипед – расчетная ячейка разбивается на элементарные ячейки. Алгоритм подразумевает произвольное задание размеров элементарной ячейки. На практике используется фиксированный размер 2,5x2,5x2,5 Å.

	Corrd(i-1,j)	Corrd(i,j)	Corrd(i,j)		
	Corrd(i-1,j)	Corrd(i,j)	Corrd(i+1,j)		
	Corrd(i-1,j)	Corrd(i,j)	Corrd(i,j)		

Заполнение структур

Зная координаты, мы знаем номер ячейки, в которую попадает атом. По каждой размерности пространства они составляют:

```
(Coordi.x - SpaceMin.x) / CellSize.x;
```

Где Coordi – координата текущего атома, SpaceMin – координата левого нижнего угла, CellSize – размер элементарной ячейки.

И наоборот, зная номер ячейки, мы знаем окружение атома по номерам соседних ячеей.

2.2.2.1 Хранение структур данных. Алгоритм модифицированных списков атомов

Для хранения структур алгоритма сканирования по пространству используется два массива.

Первый CellList1 размерности M=SpaceSize.x/CellSize.x * SpaceSize.y/CellSize.y* SpaceSize.z/CellSize.z и второй CellList2 размерности M+N.

где SpaceSize – размер расчетной ячейки,

CellSize – размер элементарной ячейки.

По сути CellList – трехмерный массив, линеаризованный для удобства копирования в память GPU. Линеаризованный индекс вычисляется по формуле:

```
CellX + CellY * CellDimX + CellZ * CellDimX * CellDimY,
```

где CellX, CellY, CellZ – координаты номера элементарной ячейки,

CellDimX= SpaceSize.x / CellSize.x,

CellDimY= SpaceSize.y / CellSize.y,

CellDimZ= SpaceSize.z / CellSize.z.

Значение элемента массива CellList1 равно индексу начала ячейки в массиве CellList2.

Массив CellList2 служит для хранения списка атомов, принадлежащих соответствующей элементарной ячейке. Для каждой элементарной ячейки хранится m+1 элемент массива, где m – количество атомов в элементарной ячейке. В первом элементе m[0] хранится количество атомов, а в остальных [1; m+1] номера атомов, принадлежащие этой элементарной ячейке.

Таблица. Структура массива CellList2.

Кол-во	Атом1	Атом2	Атом3	Кол-во	Атом4	Атом5	Атом6			

При добавлении атома в списки алгоритма сканирования по пространству необходимо найти начало ячейки по массиву CellList1, увеличить количество элементов, попадающих в нужную элементарную ячейку и записать непосредственно сам номер атома:

```
int n = CellX + CellY * CellDimX + CellZ * CellDimX * CellDimY; // определяем
линеаризованных номер элементарной ячейки
CellList2[CellList1[n]]++; // увеличиваем количество элементов
CellList2[CellList1[n] + CellList2[CellList1[n]]] = i; // записываем номер атома
```

2.2.2.2 Цикл просмотра соседних атомов

Необходимо вычислить невалентные взаимодействия между всеми атомами системы.

При реализации на центральном процессоре просматривается каждая элементарная ячейка. Зная номер ячейки, перечисляются атомы внутри текущей ячейки.

Основная сложность составляет перечислить соседние ячейки. Легче всего это сделать тремя вложенными циклами. По циклу для каждого измерения трехмерного пространства:

```
for (int Cell2X = Cell1X - ShiftX; Cell2X <= Cell1X + ShiftX; Cell2X++)
      for (int Cell2Y = Cell1Y - ShiftY; Cell2Y <= Cell1Y + ShiftY; Cell2Y++)
```

```
            for (int Cell2Z = Cell1Z - ShiftZ; Cell2Z <= Cell1Z + ShiftZ;
Cell2Z++)
```

где Cell1X, Cell1Y, Cell1Z – номер текущей элементарной расчетной ячейки,

Cell2X, Cell2Y, Cell2Z – номер элементарной расчетной ячейки – соседа.

ShiftX, ShiftY, ShiftZ – целочисленное смещение.

Смещение определяется как количество расчетных ячеек, попадающих в радиус взаимодействия и округляется в большую сторону.

При подходе к стенке тройной цикл будет выдавать номера несуществующих ячеек.

Далее в программе реализована функция CellExist, которая получает на вход номера ячейки по каждому измерению и размерность пространства. Функция возвращает «1», если такая ячейка существует, либо «0» в противном случае.

Дальнейшие вычисления невалентных сил происходят только если ячейка-сосед существует.

Тройной цикл выдает также номер текущей ячейки. Текущую ячейку необходимо просматривать дважды: чтобы найти исходные атомы и их соседей. В общем случае предполагается, что в одной элементарной ячейке могут находиться несколько атомов. Это количество можно уменьшить до одного, уменьшив размеры элементарной ячейки. Но алгоритмы пишутся для общего случая.

Алгоритм расчета сил получает на вход номера атомов. Производится две проверки:

1. Это не один и тот же атом?
2. Не содержатся ли атомы в списке валентных взаимодействий?

Разумеется, на это также тратится машинное время.

Список валентных взаимодействий моно воспринимать как список «запрещённых» взаимодействий. Эти пары атомов уже были учтены при расчете валентных взаимодействий.

2.2.3 Алгоритм составления списков Верле

Ускорение работы программы моделирования молекулярной динамики за счет реализации алгоритма составления списков на CUDA

Итак, необходимо составить список пар атомов, взаимодействующих друг с другом.

Программа на центральном процессоре выглядит тривиально

```
int NumberOfPairs=0;
for(int i=0; i<N-1; i ++)
     for(int j=i+1; i<N; j++)
{
     if (Distamce(Atom[i], Atom[j]<=15)
     {
          ListOfPairsX[NumberOfPairs]=i;
          ListOfPairsY[NumberOfPairs]=j;
          NumberOfPairs++;
     }
}
```

Пары вида x-y и y-x по сути являются одной парой. И это отражено в показанном алгоритме. Однако при реализации параллельной версии программы удобнее рассчитывать силы, действующие на атом x со стороны атома y, и не на оборот. Таким образом в список Верле должны попадать и пары x-y и y-x.

Заметим, что для выполнения цикла потребуется $N^2/2$ операций. Если система содержит 10^5 атомов, то нам потребуется 10^{10} шагов, что займет некоторое время, как показывают практические данные, превышающее в несколько раз время расчета самих сил.

Размер массивов взаимодействующих пар неизвестен. Выделять под них N^2*sizeof(int) байт памяти не всегда корректно. Как минимум, мы имеем существенное количество неиспользованной памяти. Как максимум, нам может просто не хватить памяти ЭВМ.

Из физики задачи известно, что количество пар намного меньше $N^2/2$. Атом взаимодействует лишь со своим окружением, находящимся от него на расстоянии R_{Cutoff}. Забегая вперед, необходимо добавить, что список Верле создается для всех пар, находящихся на расстоянии $R_{Verle}=R_{Cutoff}+2$. Число 2 выбрано из вычислительного опыта.

Можно предложить двухэтапный алгоритм. На первом этапе вложенный цикл считает количество пар. Далее мы выделяем только необходимое количество памяти. После этого запускаем вышеприведенный вложенный цикл еще раз.

В программе PUMA-CUDA алгоритм составления списков Верле реализован как на центральном процессоре, так и на графическом. Последние нововведения пишутся, в основном, для гетерогенной версии, как показывающей наибольшее быстродействие.

На центральном процессоре наиболее эффективен алгоритм сканирования по пространству.

Для повышения быстродействия алгоритма было предложено разработать его реализацию на GPU.

Для реализации эффективного алгоритма на GPU задача должна разделяться на независимые друг от друга этапы. Причем, если на центральном процессоре мы делим задачу на небольшое количество вычислительных нитей, например, число ядер * число узлов, то для эффективной загрузки GPU число вычислительных нитей может составлять десятки и сотни тысяч.

Особое внимание при работе с GPU необходимо уделять выделению и использованию памяти. Если мы будем исходить из предположения, что для хранения информации о контактах нам потребуется изначально $N^2/2*2*sizeof(int)$[1] байт памяти, то для системы из 100 000 атомов необходимо будет выделить $4*10^{10}$=40 ТБ, что неприемлемо. Итак, мы не знаем, сколько потребуется выделить памяти. Необходимо сначала вычислить это количество, а потом заполнить матрицы перехода.

Предложим ввести массив (вектор) DIMS(N) (от dimensions). Значение каждого компонента вектора Di соответствует количеству контактов атома N с атомами под порядковыми номерами, меньшими N.

[1] sizeof(int) – количество памяти в байтах, отводимое под целочисленный тип.

Тогда для расчета количества элементов на GPU предложим следующую подпрограмму ядра – функцию, выполняемую на устройстве (графическом процессоре) параллельным образом.

```
      __global__ static void EatAllCUDA(float *x, float *y, float *z,
int * dims, float R)
      {
            int N=blockIdx.x * blockDim.x + threadIdx.x;
            float d=0;
            float xn=x[N];
            float yn=y[N];
            float zn=z[N];
            dims[N]=0;
            for (int i=N-1; i>=0; i--)
            {
                  d=(x[i]-xn)*(x[i]-xn)+(y[i]-yn)*(y[i]-yn)+(z[i]-zn)*
(z[i]-zn);
                  if (d<=R*R)
                        dims[N]++;
            }
      }
```

Где x, y, z – массивы соответствующих координат атомов,

R – радиус обрезания, критерий, по которому атомы попадут в список.

Запуск параллельной функции осуществляется следующим образом.

```
EatAllCUDA<<<N/256, 256>>>(dev_x, dev_y, dev_z, dev_dims, R);
```

Где dev_x, dev_y, dev_z – адреса массивов с координатами в адресном пространстве устройства,

Dev_dims – адрес массива количества контактов атомов в адресном пространстве устройства,

N – количество атомов в системе.

R – радиус обрезания, критерий, по которому атомы попадут в список.

256 – количество нитей в блоке.

Сумма массива dims соответствует общему количеству контактирующих пар.

Для программиста удобно считать, что все функции запускаются одновременно параллельно, образуя N вычислительных нитей. На самом деле функций в общем случае больше, чем количество процессоров gpu. Вычислительные нити (threads) делятся сеткой (grid) на блоки (block). Полное количество нитей определяется как произведение размера сетки и размера блока, что в данном случае составляет N/256*256=N. Сетки и блоки бывают 1, 2 3-х мерные. В данном случае мы выбрали одномерную сетку и одномерный блок. От выбора размера блока зависит производительность. Размер блока выбран исходя из рекомендаций компании NVIDIA: размер блока должен делится на 64 и быть не меньше 192. Число 256 было выбрано для соответствия наибольшему число моделей графических ускорителей по критерию compute capability.

Исходя из алгоритма, каждая функция работает разное количество времени. Если вообразить, что все функции работают параллельно, то разумно предположить, что графический ускоритель, посчитав первые функции, будет простаивать. На самом деле это не так. Только нити, образующие warp (32 нити), выполняются параллельно. Графический процессор запускает нити варпами на выполнение, когда у него появляются свободные

мультипроцессоры. Таким образом запускаются первые нити, часть из них заканчивает свою работу, составив свою часть массива dims, и освобождает вычислительные ресурсы для следующей порции функций.

Массив dims посчитан. Теперь можно предложить использование зубчатого массива, чтобы описать контакты между атомами.

Таблица 2. Пример структуры зубчатого массива для хранения списка Верле.

						(a, a)			
						(a, a)			(a, a)
					(a, a)	(a, a)			(a, a)
					(a, a)	(a, a)		(a, a)	(a, a)
			(a, a)		(a, a)	(a, a)		(a, a)	(a, a)
		(a, a)	(a, a)	(a, a)	(a, a)	(a, a)	(a, a)	(a, a)	(a, a)
		(a, a)	(a, a)	(a, a)	(a, a)	(a, a)	(a, a)	(a, a)	(a, a)
Массив dims	1	2	3	2	5	7	2	…	6

Однако работа с линейными массивами более удобна.

Чтобы вычислить адрес элемента в новом линейном массиве, нужно найти префиксную сумму от массива dims:

```
(dims(1), dims(1)+dims(2), dims(1)+dims(2)+dims(3), …)
```

Выполнить эту задачу про центральном процессоре поможет следующий код:

```
for (int i=1; i<N; i++)
          dims[i]+=dims[i-1];
```

Для нахождения префиксной суммы с использованием gpu можно воспользоваться библиотекой cudpp, входящей в состав CUDA SDK (или Toolkit).

На следующем этапе нам необходимо заполнить массивы пар атомов. Код функции на GPU выглядит похожим образом с функцией заполнения массива dims.

```
     __global__ static void EatAllCUDAFill(float *x, float *y,
float *z, int *dims, int *pairs_x, int *pairs_y, float R)
     {
          int N=blockIdx.x * blockDim.x + threadIdx.x;
          int index=0;
          float d=0;
          float xn=x[N];
          float yn=y[N];
          float zn=z[N];
          for (int i=N-1; i>=0; i--)
          {
               d=(x[i]-xn)*(x[i]-xn)+(y[i]-yn)*(y[i]-yn)+(z[i]-
zn)* (z[i]-zn);
               if (d<=R*R)
               {
                    pairs_x[dims[N]+index]=i;
                    pairs_y[dims[N]+index]=N;
                    index++;
               }
          }
     }
```

Код запуска функции заполнения массива пар атомов выглядит следующим образом:

```
EatAllCUDAFill<<<N/256, 256>>>(dev_x, dev_y, dev_z,
dev_dims, dev_pairs_x, dev_pairs_y);
```

Задача решена.

Где dev_x, dev_y, dev_z – адреса массивов с координатами в адресном пространстве устройства,

dev_dims – адрес массива количества контактов атомов в адресном пространстве устройства,

dev_pairs_x, dev_pairs_y – адрес массивов пар атомов в адресном пространстве устройства размерностью ∑dims.

N – количество атомов в системе.

R – радиус обрезания, критерий, по которому атомы попадут в список.

256 – количество нитей в блоке.

Задача решена.

Остается открытым вопрос о возможности работы с числом атомов, не кратным размеру блока.

Мы уже решили задачу для N-int(N/256) числа частиц, где int – целая часть от числа. Вычислений осталось сравнительно немного. Можно предложить досчитать оставшуюся часть на центральном процессоре. Мы предлагаем использовать для этого GPU.

Создадим сетку из одного блока. В блоке будет N-int(N/256) вычислительных нитей. Таким образом код для запуска функции расчета количества пар, ровно как и для их заполнения будет выглядеть следующим образом:

```
if (N%256>0)
     EatAllCUDAResidue<<<1, N%256>>>(dev_x, dev_y, dev_z,
dev_dims, dev_pairs_x, dev_pairs_y, R, ((int)(N/256))*256);
```

Код функции ядра EatAllCUDAFillResidue отличается от EatAllCUDAFill только наличием смещения на ((int)(N/256))*256 элементов:

```
__global__ static void EatAllCUDAResidue(float *x, float *y,
float *z, int * dims, float R, int Shiftion)
{
     //int i = threadIdx.x;
     //int j = blockIdx.x;
     int N=blockIdx.x * blockDim.x + threadIdx.x+Shiftion;
     float d=0;
     float xn=x[N];
     float yn=y[N];
     float zn=z[N];
     //int i=all/w;
     //int j=all%w;
     dims[N]=0;
     for (int i=N-1; i>=0; i--)
     {
```

```
                d=(x[i]-xn)*(x[i]-xn)+(y[i]-yn)*(y[i]-yn)+(z[i]-
zn)* (z[i]-zn);
                if (d<=R*R)
                     dims[N]++;
           }
      }
```

Код функции EatAllCUDAFillResidue изменяется аналогичным образом и здесь не приводится.

2.2.3.1 Оптимизация алгоритма составления списков Верле

В реальной версии алгоритм составления списков взаимодействующих атомов (списка Верле) выглядит несколько иначе. Приведённый код приведён для удобства чтения и понимания.

Массив pairs_y не нужен. Мы и так знаем номер текущего атома. В массиве pairs_x хранятся номера всех соседей и их количество.

Составление списка Верле ведётся по структурам алгоритма сканирования по пространству. Зная номер атома, мы знаем номер элементарной расчетной ячейки. Далее мы находим все элементарные ячейки, находящиеся на некотором окружении от текущей. Зная номер ячейки, мы перечисляем атомы в ней и составляем список Верле только из тех атомов, которые находятся на заданном расстоянии от текущего атома. Атомы в текущей ячейке также просматриваются.

В список взаимодействующих атомов попадает несколько больше атомов, чем R_{Cutoff}. Список создаётся исходя из радиуса $R_{Verle}=R_{Cutoff}+2$. Как было сказано выше, число 2 было выбрано из вычислительного опыта.

Б*о*льший радиус составления списка Верле позволяет составлять список не на каждом шаге, а согласно критерию:

$$|Coord_i-PreviousCoord_i|<= (R_{Verle}-R_{Cutoff})/2$$

Где PreviousCoord – предыдущие координаты. Они обновляются не на каждом шаге, а лишь когда критерий составления списка перестаёт выполнятся. Иными словами, когда атомы отошли от зафиксированных позиций на большое расстояние.

2.2.3.2 Оптимизация радиуса попадания в список Верле

Разность $R_{Verle}-R_{Cutoff}$ влияет на частоту составления списка Верле. По умолчанию в программе выбраны $R_{Verle}=12.5$ и $R_{Cutoff}=10.5$. Если радиус экранирования (обрезания) R_{Cutoff} кулоновских и ван-дер-Ваальсовых взаимодействий продиктован чисто физическими соображениями, то от выбора радиуса составления списке $R_{Verle}=12.5$ зависит лишь быстродействие программы.

В минимальном случае (хотелось написать в худшем случае, но неизвестно, что хуже) $R_{Verle}=R_{Cutoff}$ придётся составлять список взаимодействующих атомов на каждом шаге, что вычислительно невыгодно. По другую сторону лежит выбор слишком большого радиуса попадания атомов в список. Именно в худшем случае при R_{Verle}=SpaceSize (размер системы, можно также написать знак бесконечности) в список попадут все пары атомов. В этом есть и положительный момент – список больше не нужно будет пересоставлять. Но безусловно, повышения быстродействия от такого списка ждать не придётся. Все атомы будут взаимодействовать со всеми. Это худший случай.

Выбор радиуса попадания атомов в список Верле подлежит оптимизации. Для малоподвижных систем его можно уменьшить, а для более подвижных увеличить. К тому же, на разных вычислительных установках и с различным количеством атомов оптимальный выбор радиуса попадания в список будет свой. А при изменении размеров системы опять же этот радиус придётся уточнять

В PUMA-CUDA реализован алгоритм оптимизации во время выполнения (real-time). В файле параметров задаётся RVerleDelta, который по умолчанию равен 0.1. Т.е. по умолчанию режим оптимизации радиуса Верле включен. В программе замеряется время, затрачиваемое на выполнение некоторого количества шагов, которое равно периоду вывода на экран Nterm. Спустя это время радиус списка Верле меняется на приращение RVerleDelta. И снова замеряется время выполнения. Если оно меньше предыдущего, то мы идём в сторону оптимального выбора радиуса попадания в список Верле. В противном случае меняем величину приращения на противоположную.

Рекомендуется брать достаточно большой период вывода на кран, порядке 10000 шагов. За этот период, помимо вычислений самих сил, необходимо несколько раз перезаполнить список пар взаимодействующих атомов. На это действие также уходит машинное время.

При заниженном периоде вывода на экран, что бывает полезно при интерактивной работе с большими системами, возможно биение радиуса Верле. На каком-то периоде времени будут вычисляться только силы, а на каком-то ещё будет перезаполняться список Верле, что не даст правильную оценку быстродействия программы.

При необходимости можно отключить оптимизацию выбора радиуса списка Верле, обнулив значение RVerleDelta в файле параметров:

```
RVerleDelta=0;
```

2.3 Интегрирование уравнений движения

Уравнения движения взаимодействующих материальных частиц обычно записывают в виде классических уравнений Ньютона

$$m_i \frac{d^2 r_i(t)}{dt^2} = F_i(r_i,\ldots,r_N),\ i=1,\ \ldots,\ N \qquad (1)$$

Удобно также придать им форму системы уравнений 1-го порядка:

$$\frac{dr_i(t)}{dt} = v_i,\quad \frac{dv_i(t)}{dt} = \frac{1}{m_i} F_i(r_i,\ldots,r_N),\ i=1,\ \ldots,\ N \qquad (2)$$

Здесь $v_i \equiv v_i(t)$ — скорости частиц, $F_i(r_1, \ldots, r_N) \equiv F_i(r_1,(t) \ldots, r_N(t)) \equiv F_i(t)$ — силы, действующие на частицы в момент времени *t*.

Интегрирование уравнений движения, представленных в виде (1) или (2),| осуществляется численно, посредством перехода к их разностной аппроксимации. Представим наиболее часто используемые алгоритмы [*Allen M. P., Tildesley D. J.* Computer simulation of liquids. Oxford: Claredon Press, 1987. 385 p].

Пусть X(t), V(t) и a(t)= *F(t)/m* — координата, скорость и ускорение для любой из компонент решения уравнений (1) или (2). Предлагаемые здесь разностные схемы и их свойства легко выводятся из рассмотрения разложений в ряды Тэйлора функции *X(t)* в окрестности произвольного момента времени t_0. Обозначив Л = $t - t_0$, имеем:

$$X(t_0+h)=X(t_0)+V(t_0)h+\frac{1}{2}a(t_0)h^2+\frac{1}{6}X^{(3)}(t_0)h^3+O(h^4) \quad (3)$$

$$X(t_0-h)=X(t_0)-V(t_0)h+\frac{1}{2}a(t_0)h^2-\frac{1}{6}X^{(3)}(t_0)h^3+O(h^4)$$

где O(Ti^4) означает члены порядка /r^4 и выше. Сложение строк в (3) дает выражение для координаты в последующий момент времени через ее координаты и ускорение в предыдущие моменты времени:

$$X(t_0+h)=2X(t_0)-X(t_0-h)+a(t_0)h^2+O(h^4) \quad (4)$$

Вычитание второго равенства из первого приводит к выражению дяя скорости через известные координаты:

$$V(t_0)=\frac{X(t_0+h)-X(t_0-h)}{2h}+O(h^2) \quad (4)$$

2.3.1 Скоростной алгоритм Верле

Для нахождения координат и скоростей атомов в одни и те же моменты времени на регулярном шаге используется следующая схема:

$$r_i(t+h)=r_i(t)+v_i(t)h+0,5a_i(t)h^2, \quad (5)$$

$$v_i(t+h)=v_i(t)+0,5[a_i(t)+a_i(t+h)]h.$$

Если в момент времени *t* известны координаты, скорости и ускорения атомов, то по ним находятся координаты в следующий момент времени *t + h*, затем силы и соответствующие им ускорения, а после этого уже скорости. Эта схема дает аппроксимацию на шаге решения уравнений движения порядка h^4 для координат и порядка h^3 для скоростей атомов.

Различие этих алгоритмов только в удобстве и точности нахождения скоростей. В этом смысле несомненным преимуществом при необходимости одновременного нахождения координат и скоростей частиц обладает скоростной алгоритм Верле.

2.3 Термостатирование

Температура в модельной молекулярной системе (мгновенная и средняя по траектории) определяется через среднюю кинетическую энергию, приходящуюся на одну степень свободы. В изолированной системе сохраняется полная энергия, а температура вычисляется как среднее по расчетному участку траектории. Это бывает очень неудобно при планировании вычислительных экспериментов, поскольку неизвестно заранее, какие задать начальные данные, координаты и скорости частиц и, тем самым, полную энергию системы, чтобы они соответствовали требуемой температуре.

Для достижения этой цели предлагаются разные подходы. Исходят из того, что моделируемая система не изолированная, а как-то взаимодействует с внешней средой, термостатом. При этом уравнения движения модифицируются, в них появляются дополнительные члены, ответственные за такие взаимодействия. Воздействие внешней термодинамической среды на молекулярную систему осуществляется посредством как-то устроенных дополнительных сил. В различных подходах исходят из разных физических соображений. Помимо температуры, такие взаимодействия могут также передавать и другие свойства среды, такие, скажем, как вязкость (броуновская или ланжевеновская динамика; столкновительная динамика) и атомарность среды (столкновительная динамика, метод Андерсена). Далее будем считать, что внешняя среда характеризуется

термодинамической температурой T_{ref}. Текущую (мгновенную) температуру будем обозначать T(t) или просто T и вычислять в соответствии с ее определением по формуле.

$$k_b T = \frac{2K(t)}{s}$$

Где s – полное число степеней свободы.

Столкновительная динамика относится к группе методов молекулярной динамики при постоянной температуре. Она исходит из представления среды, в которую помещена моделируемая молекулярная система, в виде виртуальных точечных частиц, имеющих распределение Максвелла по скоростям.

Распределение скоростей отвечает заданной температуре T_{ref}. Виртуальные частицы среды в случайные моменты времени упруго сталкиваются с атомами, которые при этом также рассматриваются как точечные частицы. Уравнения движения имеют вид:

$$\frac{dr_{i,\alpha}}{dt} = v_{i,\alpha}, \quad m_i \frac{dv_{i,\alpha}}{dt} = F_{i,\alpha} + \sum_k f_{ik,\alpha}\delta(t - t_{ik}),$$
$$i = 1, 2, \ldots, N; \quad \alpha = \{x, y, z\}.$$

Здесь δ(t) — дельта функция Дирака; $f_{ik,a}$ — стохастическая импульсная сила, приводящая к скачку скорости i-го атома в случайные моменты времени t_{ik} . Значение скачка скорости вычисляется как результат столкновения двух точечных частиц, имеющих перед столкновением скорости v (атом) и v_0.

$$\Delta v(t) = \frac{2m_0}{m_0 + m}\left(v_0(t) - v(t)\right)$$

Здесь m – масса моделируемой частицы, m_0 – масса виртуальной частицы. Скорости v_0 выбираются из распределения Максвелла.

$$P(\mathbf{v}) = \left(\frac{m_0}{2\pi T_{ref}}\right)^{3/2} \exp\left\{-\frac{m_0 \mathbf{v}^2}{2T_{ref}}\right\}.$$

Столкновения происходят в соответствии с процессом Пуассона, задаваемого единственным параметром λ — средним числом столкновений атома с частицами среды в единицу времени, частотой столкновений. Параметр λ столкновительной молекулярной динамики аналогичен параметру ζ в броуновской динамике. При небольшой величине λ воздействие виртуальной среды на динамику молекулярной системы будет незначительным, и тогда виртуальная среда выступает только как внешний термостат, необходимый для поддержания температуры.

2.3.1 Реализация

В файле параметров температура задаётся значением TRef секции [main]. Частота столкновений λ определяется параметром Lamd.

Текущее значение заданной температуры зависит от парамметров:

- dTRef – величина приращения заданной температуры за 1 ps;

- dTRank – изменение заданной температуры, в зависимости от номера реализации (ранка) при работе в среде MPI.

Параметр dTRef полезен для работы в режиме линейного нагревания.

Параметр dTRank удобно использовать для одновременного расчета нескольких траекторий при различных температурах.

Алгоритм поддержания постоянной заданной температуры Tref выглядит следующим образом:

1. Описанные действия выполняются отдельно с каждым атомом.
2. Столкновение атома с виртуальной частицей происходят согласно вероятности:
   a. rand() / RAND_MAX > exp(-Lamd * Tau)
3. Разыгрываем скорости виртуальной частицы:
   a. V.x = GiveRandom(S, 0);
   b. V.y = GiveRandom(S, 0);
   c. V.z = GiveRandom(S, 0);
   d. Где функция GiveRandom(S, A) выдаёт случайную величину с математическим ожиданием A и дисперсией S. Дисперсия задаётся как $\sqrt{\frac{T_{ref}}{m_0}}$
4. В том случае, если столкновение произошло, то к скоростям атома прибавляем:
   a. dV = (V - Velo[i]) * (2.0 * m0 / (m0 + m))

2.3 Периодические граничные условия. Прямоугольная расчетная ячейка

При работе в периодических граничных условиях вводится понятие расчетной ячейки – это область пространства, ограниченная прямоугольным параллелепипедом. Система должна полностью помещаться в расчетную ячейку. Один из подходов расположения расчетной ячейки и вполне естественный заключается в том, чтобы расположить параллелепипед в том месте, где расположена система. Другой подход – менее очевидный, но вычислительно более выгодный – заключается в том, чтобы один из углов ячейки располагать в начале координат. Тогда размер параллелепипеда совпадёт с его границами. Он будет располагаться в положительном направлении каждой из осей координат.

Всё пространство разделяется на ячейки. Предполагается, что моделируемая система находится в расчетной ячейке. А в остальных ячейках находятся точно такие же системы – образы настоящей системы. Либо наоборот: координаты в расчетной ячейки являются образами системы. Координаты системы и её образов в точности различаются на размеры расчетной ячейки (помноженные на n, где n – целое).

2.3.1 Вычисление координат образа системы в расчетной ячейке

Для работы с периодическими граничными условиями необходимо уметь составлять координаты в расчетной ячейке:

1. Все проекции координат x, y и z обрабатываются одинаково и независимо друг от друга (в тексте будет встречаться координата x, сказанное необходимо повторить и для y, и для z).
2. Если координаты положительны и не превышают размеров расчетной ячейки, то оставляем их без изменения.
3. Если координата по любой оси отрицательная, то координата образа рассчитывается по формуле:

   ```
   CoordImage.x+= Coord.x +SpaceSize.x - fmod(-Coord.x, SpaceSize.x);
   ```

   где CoordImage – образ координаты,

   x – любая ось (x, y, z),

   SpaceSize.x – размер расчетной ячейки по соответствующей оси,

   Coord.x – координаты системы по соответствующей оси,

   fmod – остаток от деления. Эта функция аналогична функции “%” языков C/C++, только работает с вещественными аргументами двойной точности (типа double).
4. В оставшемся случае координаты по любой оси рассчитываются по формуле:

   ```
   CoordImage[i].x = Coord[i].x+ fmod(Coord[i].x, SpaceSize.x)
   ```

   Где функция fmod – остаток от деления. Аналогичен функции «%», только работает с вещественными числами.

2.3.2 Модификация алгоритма сканирования по пространству

Благодаря удачной организации программы алгоритм сканирования по пространству подвергся лишь незначительным изменения.

По-прежнему идёт расчет взаимодействий между атомами, находящимися в рассматриваемой элементарной расчётной ячейке и её соседей. По-прежнему соседние ячейки находятся с помощью трёх вложенных циклов:

```
for (int Cell2X = Cell1X - ShiftX; Cell2X <= Cell1X + ShiftX; Cell2X++)
      for (int Cell2Y = Cell1Y - ShiftY; Cell2Y <= Cell1Y + ShiftY; Cell2Y++)
            for (int Cell2Z = Cell1Z - ShiftZ; Cell2Z <= Cell1Z + ShiftZ; Cell2Z++)
```

где Cell1X, Cell1Y, Cell1Z – номер текущей элементарной расчетной ячейки,

Cell2X, Cell2Y, Cell2Z – номер элементарной расчетной ячейки – соседа.

ShiftX, ShiftY, ShiftZ – целочисленное смещение.

Отличие заключается в том, что если номера элементарной расчетной ячейки выходят за пределы «большой» расчетной ячейки, то они всё равно существуют.

Если мы не попадаем в область расчетной ячейки, то необходимо пересчитать номер элементарной ячейки-соседа таким образом, чтобы это был номер элементарной ячейки, являющейся образом соседа текущей. Также необходимо определить вектор смещения, который перенёс бы координаты реальной ячейки в ячейку-образ.

Алгоритм пересчета для всех проекций координат выглядит похожим образом. Поэтому приводятся алгоритм для проекции X.

Отдельно рассматривается случай, когда номер элементарной ячейки вышел за пределы пространства расчетной ячейки в большую сторону и отдельно, когда оказался отрицательным.

```
if (Cell2Xi >= CellDimX)
{
	ShiftCell.x = Cell2X / CellDimX;
	Cell2Xi -= ShiftCell.x * CellDimX;
	ShiftCell.x *= SpaceSize.x + Beveled.x;
}
if (Cell2Xi < 0)
{
	ShiftCell.x = -(Cell2X + 1) / CellDimX + 1;
	Cell2Xi += CellDimX * ShiftCell.x;
	ShiftCell.x = -ShiftCell.x * SpaceSize.x + Beveled.x;
}
```

2.3.3 Модификация алгоритма составления списка Верле

В список пар взаимодействующих атомов необходимо добавить соседей, находящихся в ячейках – образах.

Добавления только лишь одноного номера атома недостаточно. Непонятно, какой атом имеется в виду: в текущей расчетной ячейке или её образе? Для устранения неопределенности вводится массив смещений. Для каждого атома сохраняется его смещение относительно его положения в расчетной ячейке. Если взаимодействующий атом находится в расчетной ячейке, то его смещение равно нулю. А если в образе расчетной ячейке, то его смещение равно размеру ячейки по соответствующей оси.

Таким образом в список Верле каждый атом может попадать по нескольку раз: как атом в расчетной ячейке и как атом в образе. У всех них будет разное вмещение.

Описанный алгоритм не накладывает никаких ограничений относительно соотношения размеров расчетной ячейки и радиуса экранирования. Также алгоритм позволяет вносить минимум модификаций в алгоритм вычисления невалентных энергий, в том числе и при работе в косоугольной ячейке.

2.3.4 Работа с плотностью системы

При включении режима работы с периодическими граничными условиями программа автоматически рассчитывает плотность. Это делается независимо от установок секции [Ro] файла параметров.

Плотность рассчитывается стандартным образом, как отношение суммы масс всех атомов на объем:

$$\rho = \frac{\sum_{i=1}^{N} m_i}{Size_x \cdot Size_y \cdot Size_z \cdot} \quad (2.3.4)$$

Здесь:

m_i – масса i-го атома;

$Size_x$, $Size_y$, $Size_z$ – размер расчетной ячейки.

Из формулы (2.3.4) следует, что изменение плотности происходит только при изменении размера расчетной ячейки. Помимо секции плотности, это возможно при включении баростата (работе в NPT-ансамбле), а также приведении системы к указанным размерам.

Плотность системы записывается в выходной файл энергетических траекторий, столбец Ro.

2.3.4.1 Приведение системы к желаемой плотности

Приведение системы к желаемой плотности возможно при установке константы Enabled секции [Ro]. Значение 1 соответствует включенному режиму, а 0 – выключенному.

Независимо от этой константы, расчет самой плотности происходит при включении режима работы с периодическими граничными условиями.

Изменение плотности возможно только за счет изменения размеров расчетной ячейки. За скорость изменения размеров отвечают константы Dx, Dy, Dz секции [Ro]. На каждом шаге интегрирования уравнений движения соответствующий размер расчетной ячейки умножается на величину (1+Dx), либо (1+Dy), либо (1+Dz):

```
SpaceSize.x *= (1 + Dx)
SpaceSize.y *= (1 + Dy)
SpaceSize.z *= (1 + Dz)
```

Выбор знака происходит автоматически. Текущий размер расчетной ячейки сравнивается с константами DestinationX, DestinationY и DestinationZ секции [Ro] файла параметров. Если текущий размер меньше заданного, то выбирается положительное приращение |Dx|, если больше заданного, что используется отрицательное приращение -|Dx|. Таким образом от установки знака констант Dx, Dy, Dz ничего не зависит.

Процесс изменения размеров расчетной ячейки останавливается при достижении заданной плотности. Ввиду того, что установление точного значения плотности кажется маловероятным, программа не меняет размер объем системы при достижении плотности:

|Ro – RoDestination| <= RoAccuracy

Здесь:

Ro – текущая рассчитанная плотность системы;

RoDestination – заданная плотность системы (устанавливается одноимённой константой секции [Ro] файла параметров);

RoAccuracy – точность, устанавливается одноимённой константой секции [Ro] файла параметров.

Типичное значение точности 0.05 кг/м^3, значения приращения плотности 0,001.

Единица измерения плотности в файле параметров и в выходных данных (файл 100ww.txt) – **кг/м^3**.

При изменении плотности, т.е. при изменении размеров расчетной ячейки, происходит аффинное растяжение пространства. Формулы выглядят достаточно просто благодаря тому, что левый нижний дальний угол расчетной ячейки помещён в начало координат.

```
Coord[i].x = Coord[i].x * (1 + Dx);
```

```
Coord[i].y = Coord[i].y * (1 + Dy);
Coord[i].z = Coord[i].z * (1 + Dz);
```
Здесь:

Coord[i] – вектор координат i-го атома;

Dx, Dy, Dz – приращение плотности на каждом шаге численного интегрирования.

Аналогичным образом выглядит при ращение образов координат:

```
CoordImage[i].x *= (1 + Dx);
CoordImage[i].y *= (1 + Dy);
CoordImage[i].z *= (1 + Dz);
```
Здесь:

CoordCoord[i] – вектор координат i-го атома;

Dx, Dy, Dz – приращение плотности на каждом шаге численного интегрирования.

2.4 Периодические граничные условия в косоугольной в расчетной ячейке

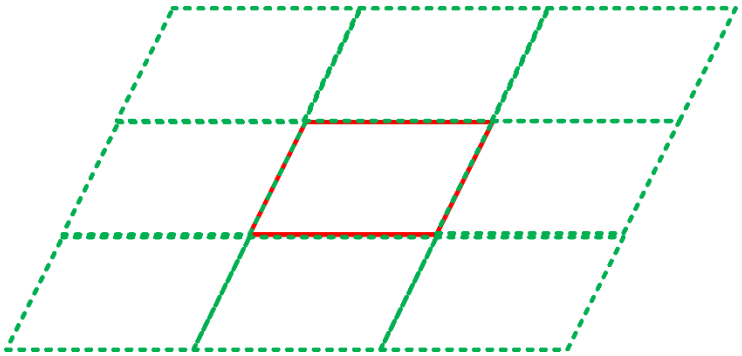


Рисунок 1. Расположение косоугольной расчетной ячейки (выделена красным) и её образов (выделено зелёным) на плоскости.

Для работы в периодических граничных условиях помимо расчетной ячейки, выполненной в форме прямоугольного параллелепипеда, используется также наклонный параллелепипед или *косоугольная расчетная ячейка*. Каждая грань такого параллелепипеда – параллелограмм, в отличии от прямоугольника у прямоугольного параллелепипеда.

Косоугольная ячейка задаётся тремя размерами по осям AX, AY и AZ, а также котангенсами углов.

Расположение размеров для проекции на плоскость XY показана на следующем рисунке:

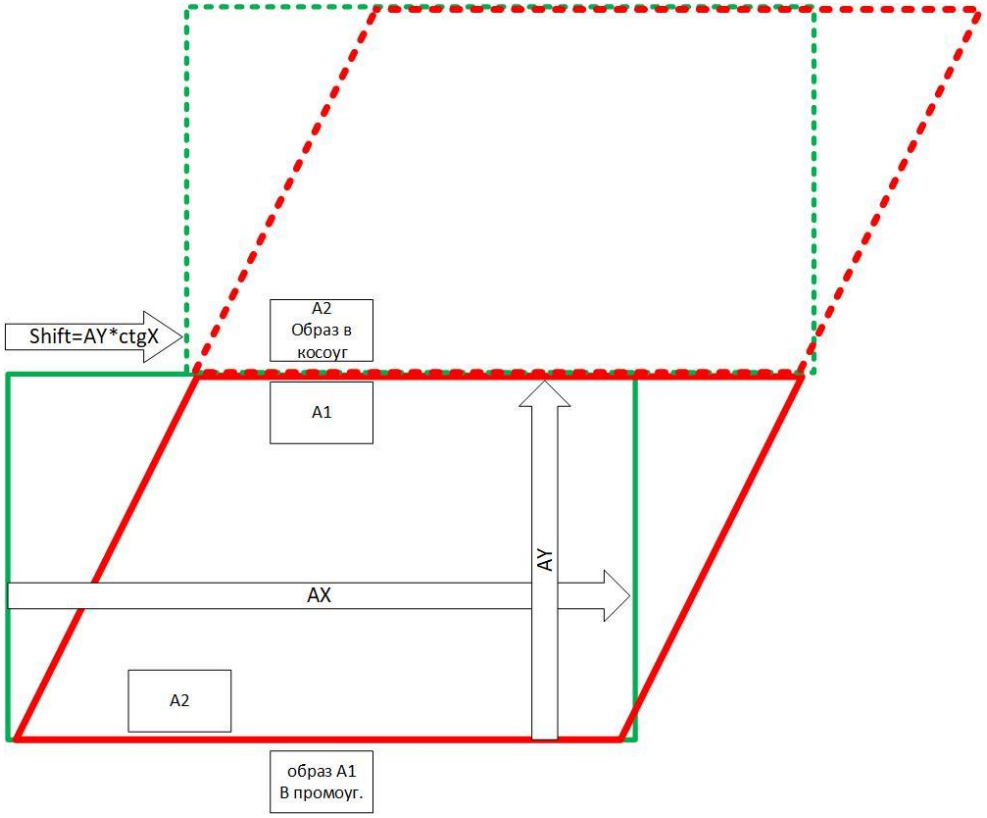


Рисунок 2. Схема сдвига координат при работе в периодических граничных условиях с косоугольной расчетной ячейкой на плоскости XY

AX – это длина стороны по оси OX, а AY – это высота параллелепипеда, а не длина стороны.

Выбор не просто углов, а их котангенсов обусловлен удобством представления:

$$ctg(\alpha) = \frac{dx}{dy}$$

$$ctg(\beta) = \frac{dx}{dz}$$

$$ctg(\gamma) = \frac{dy}{dz}$$

2.1.1 Реализация

Для реализации косоугольной расчетной ячейки в программе необходимо следовать следующим правилам:

1. Если рассматривать направление вдоль оси X (без смещения по осям Y и Z), то в периодических граничных условиях выбор в качестве ячейки прямоугольника аналогичен параллелограмму. См. рис. 4.2: красная сплошная линия – параллелограмм аналогичен выбору в качестве ячейки прямоугольника – зелёная сплошная линия. Именно поэтому по оси Y выбирается высота параллелограмма, а не длина его вершины.
2. В направлении Y идёт смещение прямоугольников на величину:

a. ShiftX=AY* $ctg(\alpha)$ для каждого нового ряда образов расчетной ячейки. Если радиус экранирования меньше размеру AY, то достаточно рассмотреть только один ряд образов.

b. В общем случае для положительно смещения по оси Y формула выглядит как:

   ShiftX=[CellY/DimY]*AY* $ctg(\alpha)$

   Где CellY – номер ячейки по оси Y в алгоритме сканирования по пространству;

   DimY – размерность массива сканирования по оси Y;

   здесь под квадратными скобками [] понимается целая часть числа

   Таким образом первый ряд образов смещается на величину AY* $ctg(\alpha)$, второй на 2*AY* $ctg(\alpha)$ и т.д.

c. В тривиальном направлении оси Y смещение определяется как

   ShiftX=-[-(CellY+1)/DimY]*AY* $ctg(\alpha)$

3. Для оси Z поступать также, как в прямоугольных периодических граничных условиях.

Ключевой момент – переход от косоугольной расчетной ячейке опять к прямоугольной, только со смещениями.

Именно благодаря ключевому моменту в алгоритмы PUMA-CUDA вносится не там много изменений, а именно:

1. Упаковка координат всех атомов в расчетную ячейку с учетом сдвига по оси Y.
2. Выбор ячеек в алгоритме сканирования по пространству с учётом смещения по оси Y.
3. Выбор атомов в алгоритме составления пар взаимодействующих атомов с учетом смещения по оси Y.

2.1.2 Упаковка координат всех атомов в расчетную ячейку

Упаковка координат всех атомов в расчетную ячейку тривиальна: из проекции x каждого атома вычитается её смещение ShiftX.

2.4.1.2 Выбор ячеек в алгоритме сканирования по пространству

При выборе несуществующей ячейки в тройном цикле (3.3) сканирования по пространству необходимо дополнить выбор:

- смещением атома по оси X на величину ShiftX
- смещение номера ячейки на

```
Cell2Xi -= int(DimY * CtgA)*int(Cell2Y / CellDimY)
```

```
для положительного смещения по оси OY
```

```
и
```

```
Cell2Xi += int(DimY * CtgA) * int(-(Cell2Y+1) / CellDimY+1);

Для отрицательного смещения по оси OY
```

Заметим, что при значении ShiftX, не кратном размеру элементарной ячейки, мы можем учесть не все атомы. Поэтому тройной цикл сканирования по ячейкам-соседям (3.3) заменяется на:

```
for (int Cell2X = Cell1X - ShiftX-1; Cell2X <= Cell1X + ShiftX+1; Cell2X++)
      for (int Cell2Y = Cell1Y - ShiftY; Cell2Y <= Cell1Y + ShiftY; Cell2Y++)
            for (int Cell2Z = Cell1Z - ShiftZ; Cell2Z <= Cell1Z + ShiftZ;
Cell2Z++)
```

Таким образом мы добавляем лишние атомы, взаимодействия для которых чуть позже не учитываем в функции расчета сил при проверки критерия дальности атомов друг от друга.

В алгоритме сканирования по пространству указанные дополнения в алгоритм расчета невалентных взаимодействий в периодических граничных условиях выполняются на **каждом шаге** интегрирования уравнений движения.

2.4.1.3 Выбор атомов в алгоритме составления списка Верле

У алгоритма составления списков пар взаимодействующих для каждой пары атомов указывается их смещение (как трехмерный вектор).

На этапе составления списка в это смещение добавляется дополнительное смещение по оси Y.

Также в алгоритм составления списка Верле вносятся изменения, аналогичные методу сканирования по пространству ввиду того, что составления списка Верле основано на алгоритме сканирования по пространству.

Составления списка пар взаимодействующих атомов работает только раз в некоторое количество шагов, когда атомы отойдут на расстояние, равное половины разности радиуса списка Верле и радиуса обрезания потенциалов невалентных взаимодействий.

Таким образом, быстродействие программы при вводе косоугольной расчетной ячейки **практически не страдает при использовании составления списка Верле**.

2.5 NPT-ансамбль. Баростат

NPT-ансамбль реализуется при помощи баростата. Программный комплекс PUMA-CUDA позволяет поддерживать постоянную температуру системы благодаря термостату и постоянное давление при помощи баростата.

Возможность работает только в периодических граничных условиях, исходя из физики задачи. Опцию баростата нельзя одновременно использовать с опцией приведения системы к заданной плотности.

Изменение давления в данном случае достигается путём изменения объема системы. Компоненты давления по осям X, Y и Z рассчитываются независимо. И баростат по трём осям работает независимо. К примеру, для «бесконечной» плоской системы можно включить баростат по осям X и Y, и не включать по оси Z.

В файле параметров баростат включается заданием значение 1 опции BarostatEnabled секции PBC (периодических граничных условий).

Также необходимо указать размер статистики, по которому усредняемся давление константой StatisticSize, которая измеряется в шагах. Как правило, достаточно 50-100 шагов.

При запуске программы выделяется массив размером StatisticSize, который служит кольцевым буфером хранения значений давления на каждом шаге статистики. На i-ом шаге интегрирования уравнений движения давление записывается в [i%StatisticSize]-ый компонент массива сбора статистики, где «%» – остаток от деления. Таким образом, хранятся всегда только StatisticSize последних значений давления.

Расчет среднего значения давления происходит на каждом шаге, как среднее значение элементов

Желаемое значение давления указывается константами PRefx, PRefy и PRefz соответственно. Единицы измерения – гигапаскали. Атмосферное давление соответствует 0,0001 ГПа = 100 000 Па. В молекулярной динамике флуктуации давления составляют именно гигапаскали.

Действие баростата заключается в изменении размеров расчетной ячейки за счет множителя bettaP. В файле параметров задаётся отдельно три множителя bettaPx, bettaPy и bettaPz для каждого измерения. Рекомендуется начинать со значения bettaP=0.0001. Величина безразмерная.

Формулы для изменения размера расчетной ячейки выглядят следующим образом:

```
SpaceSize.x *= 1 + bettaPx * PisOpen * (PMedian.x - PRefx) * h;
SpaceSize.y *= 1 + bettaPy * PisOpen * (PMedian.y - PRefy) * h;
SpaceSize.z *= 1 + bettaPz * PisOpen * (PMedian.z - PRefz) * h;
```

Здесь:

SpaceSize – вектор – размер расчетной ячейки;

bettaPx, bettaPy, bettaPz – множитель изменения размеров расчетов ячейки;

PMedian – вектор – среднее значения давления за StatisticSize шагов интегрирования;

PRefx, PRefy, PRefz – заданное значение давления;

h – шаг численного интегрирования;

PisOpen – константа, принимающее значение 0 или 1. Сигнализирует о том, что баростат включен. Не равно константе BarostatEnabled.

При запуске моделирования программа пропускает StatisticSize*5 шагов для корректного расчета среднего значения давления. Константа PisOpen при начале работы программы равна нулю, тем самым баростат выключен. По прошествии указанного количества шагов баростат включается установкой константы в значение 1.

Уравнения движения принимают вид:

```
Coord[i].x += Velo[i].x * h + FPrev[i].x * (h * h / m[i] * 0.5) + bettaPx * PisOpen * (PMedian.x - PRefx) * CoordImage[i].x * h;

Coord[i].y += Velo[i].y * h + FPrev[i].y * (h * h / m[i] * 0.5) + bettaPy * PisOpen * (PMedian.y - PRefy) * CoordImage[i].y * h;

Coord[i].z += Velo[i].z * h + FPrev[i].z * (h * h / m[i] * 0.5) + bettaPz * PisOpen * (PMedian.z - PRefz) * CoordImage[i].z * h;
```

Здесь:

Coord[i] – вектор – значение координаты i-го атома;

Velo[i] – вектор – значение скорости i-го атома;

FPrev[i] – предыдущее значение силы, действующей на i-ый атом;

h – шаг численного интегрирования;

m[i] – масса i-го атома;

bettaPx, bettaPy, bettaPz – множитель изменения размеров расчетов ячейки;

PMedian – вектор – среднее значения давления за StatisticSize шагов интегрирования;

PRefx, PRefy, PRefz – заданное значение давления;

CoordImage[i] – вектор – координаты образа i-го атома в расчетной ячейке;

PisOpen – константа, принимающее значение 0 или 1. Сигнализирует о том, что баростат включен. Не равно константе BarostatEnabled.

При включенном режиме работы с периодическими граничными условиями на каждом шаге интегрирования независимо рассчитываются как сами координаты атомов, так и их образов.

Координаты образов координат меняются схожим образом:

```
CoordImage[i].x += Velo[i].x * h + FPrev[i].x * (h * h / m[i] * 0.5) + bettaPx *
PisOpen * (PMedian.x - PRefx) * CoordImage[i].x * h;
CoordImage[i].y += Velo[i].y * h + FPrev[i].y * (h * h / m[i] * 0.5) + bettaPy *
PisOpen * (PMedian.y - PRefy) * CoordImage[i].y * h;
CoordImage[i].z += Velo[i].z * h + FPrev[i].z * (h * h / m[i] * 0.5) + bettaPz *
PisOpen * (PMedian.z - PRefz) * CoordImage[i].z * h;
```

2.6 Управляемая молекулярная динамика

2.6.1 Сфероцилиндр

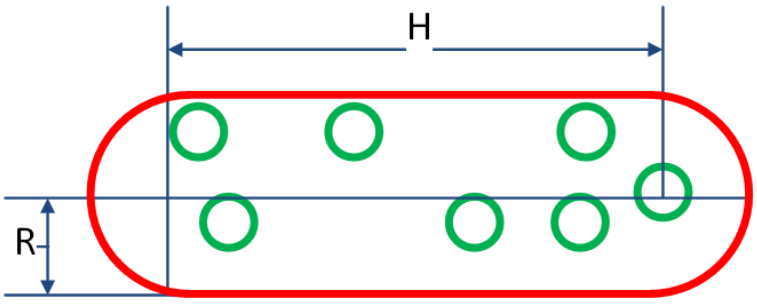


Рисунок 3. Сфероцилиндр. Красное - не пересекаемые границы, зеленое - атомы.

Моделирование системы при достаточной температуре приводит к тому, что атомы разлетаются в разные стороны на большое расстояние. По физике это были бы практически бесконечные расстояния. Как минимум, эти разлетающиеся атомы становятся неинтересны исследователю. Помимо этого, эти большие ничем не занятые области пространства приводят к увеличению памяти, необходимой для хранения структур алгоритма сканирования по пространству. Это приведёт к замедлению в работе программы, а также к исчерпанию физической и виртуальной памяти вычислительной машины. Напомним также, что алгоритм составления списков пар взаимодействующих атомов (список Верле) также использует алгоритм сканирования по пространству.

Для удержания системы в компактном виде (но не сжатом физически), предлагается установить границы пространства. В качестве границ мог бы выступать прямоугольный параллелепипед. Но в молекулярной динамике выбран сфероцилиндр.

Сфероцилиндр удобнее представить как цилиндр радиуса R, высота H которого параллельна оси OX. Дно и крышка (ввиду ориентации они находятся по бокам) не плоские, а представляют собой стенки полусферы радиуса R, центры которых совпадают с центрами граней цилиндра.

Ограничительное действие сфероцилиндра сводится к реализации отталкивающего потенциала:

$$U = \frac{\mathrm{R}_{min}{}^{12}}{(r-R)^{12}} - \frac{2\mathrm{R}_{min}{}^{6}}{(r-R)^{7}}$$

Где r – длина вектора-координаты атома, R – радиус сфероцилиндра, R_{min} – длина области отталкивающего потенциала.

Дифференцируя, получаем силу, действующую на атом:

$$F = \frac{12\mathrm{R}_{min}{}^{12}}{(r-R)^{13}} - \frac{12\mathrm{R}_{min}{}^{6}}{(r-R)^{7}} \qquad (2.6.1)$$

Сила со стороны сфероцилиндра действует в том случае, если $(r - R) \leq R_{min}$

Реализация отталкивающего потенциала сфероцилиндра состоит из двух частей: цилиндра и двух полусфер.

Частица находится **в зоне цилиндра**, если модуль её x-координаты меньше или равен половине высоты цилиндра:

$$|Coord_x| \leq \frac{H}{2}$$

В этом случае находим вектор нормали $\vec{N}$ такой, что

$N_X=0$;

$N_Y=Coord_Y$;

$N_Z=Coord_Z$.

Далее нормализуем вектор $\vec{N}$.

Вектор нормали умножаем на скалярное значение силы, получаемое по формуле (2.6.1). Это и будет сила, действующая на атом со стороны сфероцилиндра.

В случае же, если частица находится в **действии полусфер**, задача сводится к нахождению частицы в сфере, если:

- из координаты атома $Coord_X$ вычесть половину высоты цилиндра $\frac{H}{2}$ в том случае, если x-координата положительна $Coord_X>0$;
- к координате атома $Coord_X$ прибавить половину высоты цилиндра $\frac{H}{2}$ в том случае, если x-координата отрицательна $Coord_X<0$.

Далее работает похожий алгоритмz:

1. Находим вектор $\vec{N} = \overrightarrow{Coord}$.
2. Нормализуем вектор $\vec{N}$.
3. Вектор нормали умножаем на скалярное значение силы, получаемое по формуле (2.6.1). Это и будет сила, действующая на атом со стороны сфероцилиндра.

2.6.2 Воздействие силы на два атома

Рисунок 4. Растяжение молекулы за концы.

Наиболее распространённое силовое воздействие – силовое разворачивание молекулы. Как правило, силу прикладывают к концевым атомам, либо к тяжёлым концевым атомам (не к водородам).

Разделяют два типа внешних воздействий:

- Воздействие с постоянной силой;
- Воздействие с постоянной скоростью.

2.6.2.1 Воздействие с постоянной силой

К заданным двум атомам прикладывается постоянная сила. Силы, действующие на атом, равны по модулю и противоположны по знаку.

Режим включается установкой константы Enabled раздела ForceConstant значения 1 или 2. Если присваивается значение 1, то сила будет прикладываться вдоль оси OX. Если 2, то сила прикладывается вдоль оси, соединяющей атомы приложения силы.

Параметром Value задаётся абсолютное значение силы в пН. По умолчанию будет прикладываться сила в 800 пН. Если Value положительно, то сила будет растягивающей, а если отрицательно, то сжимающей.

AtomLeft и AtomRight задают номера атомов, к которым прикладывается сила.

Направления вышесказанного верны, когда проекция на ось OX AtomLeft расположена левее, чем проекция AtomRight.

AtomLeft и AtomRight считаются от 1. В языках «C/C++» индексы массивов отсчитываются от 0. Поправку расчетов номеров атомов можно делать в расчетах, а можно при чтении файла параметров. Чтобы не вызывать путаницы, в программе PUMA-CUDA пересчет номеров атомов всегда ведётся при чтении файла параметров.

Также при чтении файла параметров происходит коррекция единиц измерения в внутренние.

Эффект от силового воздействия – растяжение молекулы (либо отрыв двух структур). По траектории легко можно найти зависимость расстояния от времени при помощи программы TAMD. Для удобства в выходной файл энергетических траекторий 100ww.txt выводится расстояние между указанными атомами в графу Distance.

При выходе новых версий PUMA-CUDA в выходной файл 100ww.txt могут добавляться новые характеристики, причем не обязательно после последнего столбца. Это происходит ввиду логической группировки характеристик. Поэтому рекомендуется при обработке выходного файла, в т.ч. программой TAMD, обращаться к интересуемым колонкам по именам, а не по их индексам.

2.6.2.2 Растяжение молекулы с постоянной скоростью

Рисунок 5. растяжение молекулы с постоянной скоростью.

Растяжение молекулы с постоянной скоростью введено с целью замера силовой реакции объекта при приложении сил, приводящих к движению с постоянной скоростью указанных атомов.

С постоянной скоростью движутся не указанные в файле параметров атомы, а подразумеваемые материальные точки, кронштейны. Это кронштейны соединены пружинами жесткости K с атомами AtomLeft и AtomRight. Движение кронштейнов приводит к несущественно запоздалому движению атомов. Жесткость пружин рекомендуется задавать в районе 100 пН/Å, что сопоставимо с жесткостью валентной связи.

За работу режима отвечают сразу две секции файла параметров: ForceConstant и ForceV. В секции ForceConstant используются переменные AtomLeft и AtomRight, ответственные за номера атомов. Растяжение с постоянной скоростью включается установкой значений 1-4 в поле Enabled секции ForceV. При этом значения соответствуют следующим режимам:

- 1 – соответствует включению сил, приложенным к атомам. Движение кронштейнов происходит вдоль оси OX.
- 2 – включает также режим аффинного растяжения пространства между кронштейнами.
- 3 – растяжение ведётся вдоль линии, соединяющей указанные атомы, без аффинного растяжения пространства.
- 4 – растяжение ведётся вдоль линии, соединяющей указанные атомы (как и режим 3), но с аффинным растяжением пространства.

Сами кронштейны движутся с постоянной скоростью 0,5*Value (секции ForceV) в противоположные скорости, если Value положительно, и навстречу друг другу, если Value отрицательно. Таким образом, достигается скорость Value [Å/пс].

Координаты кронштейнов при режимах запуска Istart=-1 и Istart=0 (секции main) совпадают с позициями концевых атомов.

При продолжении моделирования (режим Istart=1) координаты кронштейнов восстанавливаются из .lst-файла.

При записи .lst-файла координаты кронштейнов записываются в последнюю строку файла через пробел в последовательности WallLeft.x, WallLeft.y, WallLeft.z, WallRight.x, WallRight.y, WallRight.z.

2.6.2.3 Аффинное растяжение пространства при растяжении с постоянной скоростью

Аффинное растяжение пространства заключается в увеличении координат всех атомов пропорционально расстоянию между кронштейнами. При этом атомы, находящиеся близко к концевым атомам, на каждом шаге получают приращение координат ±0,5*Value*tau (где tau [пс] – шаг численного интегрирования). Атомы, находящиеся ровно на середине между концевыми атомами, получают нулевое приращение.

В случае с растяжение пространства вдоль оси OX (режим Enabled=2) приращение x-координат происходит по формуле:

```
Coord[i].x += (Coord[i].x - Center.x) / (WallRight.x - Center.x) * Value * Tau * 0.5;       (формула 2.6.2.3)
```

Где Coord[i] – вектор координат i-го атома,

Center – координаты центра между концевыми атомами,

WallRight – координата правого кронштейна,

Value – константа скорости растяжения файла параметров (секции ForceV),

Tau – шаг численного интегрирования.

Заметим, что WallRight.x – Center.x = Center.x – WallLeft.x.

При растяжении вдоль линии, соединяющей два атома, аффинное растяжение пространства выглядит чуть сложнее:

```
r0 = Coord[i];
r1 = WallRight;
s = WallRight - WallLeft;
r1r0 = r1 - r0;
r2 = r1 - s * (s.CrossScalar(&r1r0) / s.CrossScalar(&s));
Coord[i].x += (r2.x - Center.x) / (WallRight.x - Center.x) * Value*Tau*0.5;
Coord[i].y += (r2.y - Center.y) / (WallRight.y - Center.y) * Value*Tau*0.5;
Coord[i].z += (r2.z - Center.z) / (WallRight.z - Center.z) * Value*Tau*0.5;
```

Здесь:

Coord[i] – вектор – координата i-го атома;

WallRight и WallLeft – вектора – координаты правого и левого кронштейнов;

Center – вектор – координаты средней точки Center = (WallRight + WallLeft) / 2;

Value – константа жесткости пружины;

Tau – шаг численного интегрирования.

Для каждого атома находится его проекция на плоскость, проходящей через правый кронштейн. Направление плоскости задаётся нормалью, проходящей через левый и правый кронштейны. Далее приращение каждой проекции координат атома рассчитывается по формуле 2.6.2.3.

2.6.3 Работа двумя подсистемами

2.6.3.1 Обозначение двух подсистем

Режим работы с двумя подсистемами задается в секции [TwoSystemsInteraction] файла параметров. Традиционно режим включается при установке константы Enabled в значение 1.

Подразумевается, что в моделируемой системе существуют отдельно две подсистемы. Каждая задаётся интервалом атомов. Первая система начинается с атома FirstAtomLeft и заканчивается FirstAtomRight, включительно. Вторая подсистема начинается с атома SecondAtomLeft и заканчивается SecondAtomRight, включительно. Соответствующие константы располагаются в секции [TwoSystemsInteraction] файла параметров.

Обязательных требований к интервалу атомов два. Первое: указанный промежуток должен существовать в моделируемой системе. И второе: номер правой границы атомов должен быть больше или равен левой.

При установке одного и того же значения в качестве левой и правой границы получается подсистема из одного атома. Это допустимо.

Первая и вторая подсистемы могут пересекаться. Это не приведёт к ошибке. Однако сложно придумать задачу, где требовались бы пересекающиеся границы. Особенно это касается случая включения силовых воздействий на подсистемы.

При включении режим работы с двумя подсистемами программа рассчитывает энергию взаимодействия между этими подсистемами как сумму энергий невалентных взаимодействий межу атомами из соответствующих подсистем. Раздельно считается сумма для кулоновского и Ван-дер-Ваальсового взаимодействий. Эти энергии записываются в выходной файл энергетических траекторий 100ww.txt в столбцы EP12qq и EP12vw. Также отдельно считаются энергии первой и второй подсистем. Они записываются в столбцы EP1 и EP2.

$$E_{12} = \sum_{i=FirstAtomLeft}^{FirstAtomRight} \sum_{j=SecondAtomLeft}^{SecondAtomRight} E_{i,j} \quad \text{(формула 2.6.3.2)}$$

2.6.3.2 Реализация расчета сил

При реализации формула 2.6.3.2 вызвала излишнюю вычислительную нагрузку. Как правило, все атомы одной подсистемы не взаимодействуют со всеми атомами второй подсистемы. Поэтому при программной реализации были введены лишь добавления расчета энергии взаимодействия в алгоритм расчета невалентных взаимодействий. В свою очередь, алгоритм расчета невалентных взаимодействий рассматривает только пары атомов, находящиеся на расстоянии радиуса экранирования.

Расчет энергий взаимодействия между двумя подсистемами потребляет некоторое количество процессорного времени. Именно по этой причине при реализации алгоритма расчета невалентных взаимодействий на графических ускорителях существуют по две функции расчета сил: одна с учетом работы с двумя подсистемами, а другая – без учета. Экономичнее и эстетичнее было бы обойтись оператором ветвления, который бы включал и выключал расчет энергий подсистем. Но этот оператор выполнялся бы в каждом вычислительном потоке, что негативно отразилось бы на производительности. Именно поэтому было принято решение оставить две похожие функции в программе.

2.6.3.3 Силовые воздействия, применяемые к двум подсистемам

В программе возможно применение силовых воздействий к двум подсистемам. Они направлены на удержание одной системы за центр масс и силовое воздействие на центр масс второй системы.

Силовые воздействия работу при включенном параметре Enabled секции [TwoSystemsInteraction] файла параметров и выбранном режиме, который задаётся параметром ForceVToSecondSystemType. Нулевое значение ForceVToSecondSystemType означает отсутствует дополнительных силовых воздействий. Ненулевое значение параметра ForceVToSecondSystemType в сочетании с нулевым Enabled также не приведёт к включение внешних воздействий.

Список силовых воздействий к двум подсистемам:

1. Удаление/приближение вдоль оси X (ForceVToSecondSystemType=1).
2. Удаление/приближение вдоль прямой, соединяющих центры масс систем (ForceVToSecondSystemType=2).
3. Удаление/приближение вдоль вектора, заданного параметрами (DirectionX, DirectionY, DirectionZ) (ForceVToSecondSystemType=3).
4. Применение постоянной силы вдоль прямой, соединяющих центры масс систем (ForceVToSecondSystemType=4). Значение силы задается параметром F.
5. Изменение четвёртой координаты второй подсистемы (ForceVToSecondSystemType=5, см. раздел «Работа с четвёртым измерением»).

Пункты 1-4 относятся на воздействие к центру масс. У первой и второй подсистем есть материальная точка – кронштейн. У первой подсистемы она покоится. У второй движется со скоростью, сказанной в константе Speed раздела [TwoSystemsInteraction]. Направление движения второй точки выбирается согласно значению константы ForceVToSecondSystemType.

Кронштейн связан с центром масс подсистемы пружиной жесткости K (одноимённая константа раздела [TwoSystemsInteraction]. Сила, действующая со стороны пружины, применяется к каждому атому согласно соотношению:

$$\overrightarrow{F_i} = K \cdot \frac{\left(\overrightarrow{\text{WallLeft}} - \overrightarrow{\text{PhysCenter1}}\right)}{M_1} * \cdot \text{m}_i$$

$$\overrightarrow{F_i} = K \cdot \frac{\left(\overrightarrow{\text{WallR}ight} - \overrightarrow{\text{PhysCenter2}}\right)}{M_2} * \cdot \text{m}_i$$

Где:

K – коэффициент жесткости пружины, соединяющий кронштейны и центры масс подсистем;

WallLeft и WallRght – вектора координат левого и правого кронштейнов соответственно;

PhysCenter1 и PhysCenter2 – вектора координат центров масс первой и второй подсистем;

M_1 и M_2 – суммарные массы подсистем;

m_i – масса i-го атома.

$$M_1 = \sum_{i=FirstAtomLeft}^{FirstAtomRight} m_i$$

$$M_2 = \sum_{i=SecondAtomLeft}^{SecondAtomRight} m_i$$

Силовое воздействие применяется пропорционально массе каждого атома подсистемы.

Параметр ForceVToSecondSystemType управляет направлением движения второго кронштейна: по прямой OX, либо по прямой, соединяющей центры масс, либо по вектору Direction, координаты которого указаны в секции [TwoSystemsInteraction] файла параметров.

2.6.4 Работа с произвольным количеством подсистем. МД-конструктор.

Работа с произвольным количеством подсистем реализована в программном продукте с целью применения сил, нацеленных на различного радо удержание групп атомов.

Применяя подобные силовые воздействия возможно придание структуре линейной или хаотичной группировке атомов.

МД-конструктор управляется установкой констант секции [Holding] файла параметров. Включение происходит заданием ненулевого значения параметра N, отвечающего за количество подсистем.

Под удержанием подразумевается:

1. Удержание группы атомов в одной плоскости (значение параметра Type=0).
2. Фиксация координат группы атомов (значение параметра Type=1).
3. Установка расстояний между атомами (значение параметра Type=2).
4. Установка потенциальной стенки-плоскости (значение параметра Type=3).

Группа атомов задается номером первого атома (параметр I1), интервалом между атомами (параметр Period) и количеством атомов (параметр Size).

Если подразумеваются пары атомов (при установке расстояний между атомами), то необходимо задавать параметры пар: I2, I2Period. Также в параметре I2L задаётся расстояние между парами атомов.

2.6.4.1 Удержание группы атомов в одной плоскости

Возможность реализована для того, чтобы группа атомов могла свободно перемещаться в одной плоскости.

Плоскость задаётся точкой R (параметр R0 секции [Holding], в котором назначаются три координаты точки через пробел) и вектором нормалью к плоскости (параметр E секции [Holding], в котором назначаются три координаты вектора через пробел).

Опция включает потенциал следующего вида:

$$U = \frac{1}{2} \cdot En \cdot (\overrightarrow{R_i} \cdot \vec{e} - \overrightarrow{R_0} \cdot \vec{e})^2$$

2.6.4.2 Фиксация координат группы атомов

При включении режима фиксации координат группа атомов остаётся неподвижной, независимо от сил, которые на них действуют.

Реализовано через обнуление скоростей. Таким образом, атомы вносят вклад в расчет потенциалов, в которых они участвуют, а также в силы, действующие от них на другие атомы.

2.6.4.3 Установка расстояний между атомами

При включении данного режима добавляются пружины, соединяющие указанные пары атомов. Исходная длина пружины задаётся параметром I2L секции [Holding], а жесткость пружина параметром En.

Опция полезна при сборке систем. Изначально можно задать малое значение En, а затем постепенно его увеличивать. Тем самым система займёт требуемую позицию.

2.6.4.4 Плоскость-стенка

Опция добавляет в систему плоскость (стенку) с потенциалом Леннард-Джонса 6-12:

$$U(r) = En \cdot \left(\left(\frac{R_{min}}{r}\right)^6 - 2\left(\frac{R_{min}}{r}\right)^{12}\right) \text{ при } r<R_{взаимодействия}$$

Rmin соответствует одноименному параметру, а $R_{взаимодействия}$ соответствует параметру Rint (сокращение от interaction), которые задаются один раз на всю секцию [Holding].

Потенциальная плоскость действует на атомы от I до I+(ISize-1)*IPeriod. Плоскость задаётся вектором-нормалью E и точку, через которую эта стенка проходит R0 (имена параметром в секции [Holding]). Эти параметры задаются столько раз, сколько необходимо установить потенциальных плоскостей. Их количество соответствует параметру N секции [Holding].

Вектор нормали плоскости не обязательно должен быть единичным. В ходе вычислений вектор будет нормирован.

Пример фрагмента файла параметров:

```
[Holding]
N=1; Задаём одну потенциальную плоскость
En=10.1; Константы при потенциале Леннард-Джонса
Rmin=1;
Rint=15;
Type=3; Ставим тип дополнительного воздействия 3 – потенциальная плоскость
```

```
I1=1; Стенка действует на атомы с 1
IPeriod=1;
ISize=1500; по 1500 с периодом 1 (каждый атом)
R0=30 0.0 0.0; Точка, через которую проходит потенциальная плоскость
E=0.0 1.0 0.0; Вектор нормали потенциальной плоскости
```

2.6.4.5 Реализация

Реализация дополнительных силовых воздействий с произвольным количеством подсистем написана для работы на центральном процессоре. Сложность задачи мала – порядка n, где n – количество затрагиваемых атомов.

Опции работы с произвольным количеством подсистем влияют на силы или скорости частиц после расчета таковых основными алгоритмами работы программы – алгоритмами расчета валентных и невалентных взаимодействий.

2.7 Четырёхмерная молекулярная динамика

2.7.1 Использование четвертого измерения для подсистемы

Предполагается, что основная система находится в привычном трёхмерном пространстве. В четырёхмерных координатах её атомы записываются как (x, y, z, 0). Т.е. все атомы основной системы по четвертому измерению имеют координату 0.

Выделяется особая подсистема, которая располагается в четвертом измерении (подсистема из четвертого измерения). Правильнее говорить, что у этой подсистемы есть ненулевая четвертая координата. Изначально все атомы из подсистемы четвертого измерения имеют вид (x, y, z, InitD4), где InitD4 – одинаковое числе в ангстремах для всей подсистемы.

Цель введения четвертого измерения для подсистемы – возможность плавного помещения подсистемы в привычное трёхмерное пространство.

В файле параметров PUMA-CUDA предусмотрен параметр InitD4 секции [TwoSystemsInteraction], который задаёт начальную четвёртую координату в четырёхмерном пространстве для второй подсистемы. Если номер атома попадает в границы [SecondAtomLeft, SecondAtomRight], то этому атому приписывается четвертая координата.

Для включения режима использования четвертой координаты необходимо выбрать пятый режим работы подсистем, установив параметр ForceVToSecondSystemType=5 секции [TwoSystemsInteraction].

По умолчанию InitD4=15 Å. Рекомендуется брать этот параметр больше радиуса экранирования невалентных взаимодействий. Таким образом подсистема из четвертого измерения не будет видная для основной системы.

Плавное приближение подсистемы из четвертого измерения в основной системе осуществляется благодаря измерению четвертой координаты подсистемы с постоянной скоростью:

Coord4 = InitD4+Speed* Tau

Где:

- Speed – скорость движения по четвёртой координате;
- Tau – шаг численного интегрирования.

Параметр Speed также находится в секции [TwoSystemsInteraction] и задаёт скорость движения в подсистемы в Å/пс. Предполагается, что для положительной начальной координаты InitD4 скорость Speed будет отрицательной.

Движение по четвертой координате прекращается, когда четвертая координата будет равно 0. В программной реализации переменная может никогда не стать в точности равной нулю. Поэтому принят допуск в Speed* Tau * 2 Å. Если четвертая координата станет меньше этого допуска, то движение прекратится.

Четвертая координата учувствует только в невалентных взаимодействиях между подсистемами. Присутствующие в подсистеме из четвертого измерения валентные связи и углы рассчитываются также, как бы они находились в третьем измерении.

При перезапуске программы в режиме продолжения значение четвертой координаты восстанавливается из .lst-файл. Она пишется в переменную, предназначенную для четвертой x-координаты правой стенки.

Алгоритм работы с четвертой координатой реализован только для периодических граничных условий. Поддерживается работа как на многоядерных процессорах, так и на графических ускорителях в режиме составления списка Верле. Алгоритм составления списка Верле не учитывает четвертую координату.

2.7.2 Полноценная четырёхмерная молекулярная динамика

Полноценная четырёхмерная молекулярная динамика подразумевает полноценную динамику по четвертому измерению.

Предполагается, что все атомы системы имеют вид (x_i, y_i, z_i, s_i), где s – четвертая координата, i=1..N, – число атомов в системе.

В начальный момент времени s_i=0 для всех атомов. Привычные первые три компоненты атомов соответствуют начальным положениям атомов, которое берётся либо из структурного файла (.str) при первом запуске программы, либо из .lst-файла при продолжении расчетов (Istart=0 или Istart=1).

Все валентные взаимодействия учитывают трехмерные координаты атомов. Невалентные взаимодействия учитывают все четыре координаты атомов.

Интегрирование уравнений движения идёт в т.ч. и по четвертой координате.

Термостат также оказывает влияние на четвертую координату. Именно благодаря термостату четвертая координата изначально отклоняется от нуля.

Алгоритм работы с четвертой координатой реализован как для изолированных систем, так и для систем с периодическими граничными условиями. Поддерживается работа как на многоядерных процессорах в режиме сканирования по пространству. Алгоритм сканирования по пространству не учитывает четвертую координату (хотя, это представляется возможным).

Запись четвертой координаты ведётся в файл траектории в первую расширенную компоненту, независимо от включения режиме записи расширенных траекторий. Остальные расширенные компоненты последуют далее за четвертой координатой при включении параметра WriteTrjExt=1 секции [main].

2.7.2.1 Введение дополнительных сил для четвертого измерения

Как и в случае с выделенной подсистемой предполагается, что четвертая координата (s) основной системы будет находится вблизи нуля (в случае выделенной подсистемы она равнялась в точности нулю).

Четвертая координата выделенной подсистемы в начальный момент времени будет равна значению InitD4.

Четвертая координата будет подвижной. На прямой OS будут работать невалентные взаимодействия, а также учитываться действие со стороны термостата. Подобно привычным трём координатам, по четвертому измерению будет происходить интегрирование уравнений движения.

На четвертую координату **основной системы** будет действовать пружина жесткостью K из секции [TwoSystemInteractions] файла параметров. Пружина прикреплена к кронштейну (стенке) с координатой (0, 0, 0, 0), либо, что соответствует реализации, в точки 0 прямой OS.

На четвертую координату **подсистемы** будет действовать пружина жесткостью K из секции [TwoSystemInteractions] файла параметров. Пружина прикреплена к кронштейну (стенке) с подвижной четвертой координатой:

Coord4 = InitD4+Speed* Tau

Где:

- Speed – скорость движения по четвёртой координате;
- Tau – шаг численного интегрирования.

Параметр Speed также находится в секции [TwoSystemsInteraction] и задаёт скорость движения в подсистемы в Å/пс. Предполагается, что для положительной начальной координаты InitD4 скорость Speed будет отрицательной.

На подсистему действует сила:

```
F.s = (Coord4-Coord[i].s) * K;
```

На основную систему действует сила:

```
F.s = - Coord[i].s * K;
```

Где F.s – четвертая компонента силы;

Coord4 – координата стенки (стенка имеет только одну координату в четвертом измерении);

Coord[i].s – четвертая компонента i-ой координаты;

K – коэффициент жесткости каждой пружины.

Таким образом, подсистема будет двигаться к основной системе с постоянной скоростью под действием пружины. Основная система, в свою очередь, будет располагаться вблизи нуля по четвертой координате под действием своей пружины.

Когда координата стенки будет подсистемы придёт в 0, то движение прекратиться.

2.7.2.2 Программная реализация

В ходе программной реализации алгоритмов работы с четвертой компонентой ставилось две задачи:

1. Не должно существовать отдельной ветви программного комплекса PUMA-CUDA, поддерживающего работу с четвертой координатой.
2. Введение четвертой координаты не должно повлиять на быстродействие программы при работе со стандартными трехмерными системами.
3. Вмешательство в программу должно быть минимальным.

Учитывая поставленные задачи, было принято решение ввести четвертое измерение благодаря модификации класса MVector, от вещающего за хранение и обработку координат одного атома. Четвертая координата была введена с помощью директивы условной компиляции USE_MD_4D.

Листинг 2.7.1 Использование директив условной компиляции при объявлении класса MVector.

```
#define USE_MD_4D
class MVector
{
public:
     MVector();
     MVector(double x, double y, double z = 0);
     //virtual ~MVector();
…
     double x, y, z;
#ifdef USE_MD_4D
     double s;
     MVector(double x, double y, double z, double s);
     double Distance4d();
     double Distance4dFrom(const MVector* V);
#endif //
}
```

Таким образом, при потребности в четвертой координате программа компилируется с включенной (раскомментированной) директивой условной компиляции USE_MD_4D.

Таким же образом вводятся дополнительные функции, присущие работе только с четырёхмерными координатами:

- MVector(double x, double y, double z, double s) – конструктор с четырьмя параметрами;
- double Distance4d() – длина четырёхмерного вектора в декартовом пространстве;
- double Distance4dFrom(const MVector* V) – расстояние между двумя четырёхмерными вектора в декартовом пространстве.

Надо заметить, что конструктор по умолчанию оставил бы четвёртую координату неинициализированной. Просто присвоить 0 переменной-члену класса s нельзя. Это переменной не существует в трехмерной версии программы. Чтобы решить эту проблему,

модифицируем конструктор по умолчанию также с помощью директивы условной компиляции:

Листинг 2.7.2 Использование директив условной компиляции при реализации конструктора по умолчанию класса MVector.

```
MVector::MVector(double x, double y, double z)
{
     this->x = x;
     this->y = y;
     this->z = z;
#ifdef USE_MD_4D
     s = 0;
#endif // USE_MD_4D
}
```

При работе на графическом ускорителе также используется класс MVector. При потребности в копировании координат атомов, к примеру, функция sizeof(MVector) вернёт разные значения в зависимости от того, используется ли трехмерная молекулярная динамика или четырёхмерная. При этом для трехмерной МД не будет передаваться лишний объем информации.