КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ВИСОКИХ
ТЕХНОЛОГІЙ

Завідувач кафедри молекулярної біотехнології та біоінформатики
доц. Олексій Юрійович Нипорко
Протокол №___ засідання кафедри
від “___ ” _____20___ р.

УДОСКОНАЛЕННЯ МЕТОДІВ РОЗРАХУНКУ ЕНЕРГЕТИЧНИХ ПАРАМЕТРІВ ДЛЯ ОКРЕМИХ ЧАСТИН СТАТИСТИЧНИХ АНСАМБЛІВ

Випускна кваліфікаційна робота бакалавра
студента спеціальності 091 «Біологія»
ОП «Біологія (високі технології)»
Головченка Максима Олександровича

Науковий керівник
завідувач кафедри молекулярної
біотехнології та біоінформатики
к. б. н., доцент **Нипорко Олексій Юрійович**

Оцінка захисту роботи

_____

Київ – 2024 р.

АНОТАЦІЯ

Головченко М.О. Удосконалення методів розрахунку енергетичних параметрів для окремих частин статистичних ансамблів. Випускна кваліфікаційна робота бакалавра за спеціальністю 091 «Біологія» ОП Біологія (високі технології).

Симуляція Молекулярної динаміки (МД) — це обчислювальний метод, у якому часова еволюція набору атомів та/або молекул, які взаємодіють між собою, прогнозується шляхом інтегрування їхніх рівнянь руху. Цей метод став досить важливим інструментом для біології, який дає можливість вивчати взаємодію та поведінку біологічних макромолекул, а також шукати та розробляти нові лікарські засоби, які націлено взаємодіють зі своєю мішенню.

Методи МД дають можливість розраховувати різні енергетичні величини, такі як потенціальна й кінетична енергія, ентальпія, ковалентні, нековалентні та кулонівські взаємодії, які дають можливість оцінити зміни які відбулися в досліджуваній системі.

Нині більшість доступних на сьогоднішній день програмних пакетів для розрахунку та аналізу МД реалізовують розрахунок більшості енергетичних величин для усієї досліджуваної системи. Проте інколи виникає необхідність в розрахунку енергетичних величин для певної частини системи, яка може являти собою певну групи атомів з яких побудована одна чи більше молекул.

В даній роботі була проведена реалізація можливості розрахунку деяких, раніше недоступних, енергетичних величин для певної частини системи у поширеному програмному пакеті GROMACS.

Ключові слова: молекулярна динаміка; енергії мікростанів; енергії окремих груп атомів.

ЗМІСТ

ВСТУП

GROMACS – це безкоштовне програмне забезпечення з відкритим вихідним кодом [1], яке містить велику кількість інструментів для підготовки досліджуваної системи, розрахунку МД та її аналізу. В основному призначений для моделювання білків, ліпідів та нуклеїнових кислот.

Головним рушієм МД в цього програмного пакету є mdrun. Саме в цьому модулі реалізовані можливості для інтегрування руху атомів разом з застосуванням допоміжних алгоритмів для моделювання умов певного статистичного ансамблю.

При розрахунку МД mdrun також розраховує, і зберігає для подальшого аналізу, різні енергетичні величини, необхідних для інтегрування руху атомів, такі як енергія ковалентних, нековалентних, кулонівських взаємодій та потенціальна енергія. Крім цього також можуть бути розраховані такі величини як кінетична енергія та ентальпія які можуть бути корисними для оцінки динаміки системи.

В доступних на сьогодні версіях GROMACS в mdrun реалізована можливість зберігати енергії нековалентних та кулонівських взаємодій для окремої частини чи частин системи, тоді як всі інші енергетичні величини зберігаються розрахованими для всієї системи. Можливим обходом цього обмеження є використання наявних в GROMACS інструментів для створення файлів топології і траєкторії, що міститимуть лише частину, обраних, атомів системи, з наступним використанням можливості mdrun читати наявні координати в траєкторії та перерахувати енергії для групи атомів, які знаходяться у файлах, створених у попередньому кроці. Проте цей спосіб не є ідеальним, оскільки mdrun для наявної траєкторії не перераховує певну частину величин, які б розраховувались при повній МД. В цьому випадку зокрема не розраховується кінетична енергія та ентальпія.

Метою цієї роботи є уможливлення розрахунку більшої кількості енергетичних параметрів з наявної траєкторії, а також спрощення процедури вибору групи атомів для перерахунку енергій.

РОЗДІЛ 1
Огляд Літератури

1.1 Принцип роботи МД

Основою для МД є точне визначення міжатомних та міжмолекулярних взаємодій. Ці взаємодії дають можливість розрахувати можливу сукупну поведінку молекулярної системи як функцію від часу. Симуляція МД використовує класичну Ньютонівську механіку для розрахунку руху атомів і, відповідно, молекул системи, на які впливають сили взаємодії між ними. Напрямок сили F, яка діє на атоми внаслідок їх взаємодії між собою, розраховується як негативний градієнт потенціалу взаємодій, що далі використовується для визначення нових позиції, яких набули атоми за певний короткий проміжок часу (зазвичай 1-2 фс, 10^{-15} с) шляхом інтегрування другого закону Ньютона $(F=am)$. Для МД використовують алгоритми чисельного інтегрування, оскільки сили, які діють на певний атом, залежать від положення інших атомів. Цей процес, в якому ітераційно знаходяться нові сили та положення атомів, використовується для отримання траєкторії, яка зберігає в собі координати та іншу інформацію про мікростан системи в певні моменти часу. Для забезпечення стабільності чисельного інтегрування, часові кроки в моделюванні МД повинні бути короткими, як правило, лише кілька фемтосекунд (10^{-15} с) кожен. Більшість важливих подій, наприклад, функціонально важливі структурні зміни в біомакромолекулі, відбуваються в часових масштабах порядку наносекунд, мікросекунд або більше. Таким чином, типове моделювання включає мільйони або мільярди кроків інтегрування. Цей факт у поєднанні з великою кількістю міжатомних взаємодій, призводить до потреби у великої кількості обчислюваних ресурсів для моделювання поведінки великих систем.

Симуляція МД може бути здійсненна за умов стандартних термодинамічних ансамблів, наприклад канонічний де кількість частинок, об’єм та температура є сталими значеннями або ізобарно-ізотермічний де, на відмінно від попереднього ансамблю, сталим підтримується тиск, замість об’єму.

Взаємодія між атомами (або групами атомів) може бути описана на квантово-механічному рівні, або за допомогою молекулярної механіки. Квантова механіка необхідна для дослідження явищ пов’язаних з електронами, зокрема перенесення електронів з утворенням та розривом ковалентних зв’язків у хімічних реакціях або поглинання електромагнітного випромінювання. Такі розрахунки є надзвичайно дорогими та наразі обмежені невеликими системами, які складаються з кількох сотень атомів, для часу моделювання менше наносекунди. У випадку молекулярної механіки застосовуються наближення, при не відбувається змін у ковалентних зв’язках, через що зберігається цілісність молекул системи. Молекулярна структура характеризується моделлю, де атоми розглядаються як сфери, які з’єднані між собою пружними зв’язками. Для обчислення значень потенціальної енергії молекулярної системи та пов’язані з нею міжмолекулярні взаємодії, використовуються набір визначених параметрів (встановлених емпірично чи за допомогою квантової механіки) та функцій, які загалом називають силовим полем. Ці функції описують потенціал взаємодії залежно від розташування атомів один відносно одного.

Функція потенціальної енергії (формули 1.1) враховує ковалентні та нековалентні взаємодії. Для ковалентних взаємодій враховуються зміни їхньої довжини та кутів між ними (формули 1.2). Також враховуються кути обертання навколо зв’язків для відтворювання правильних ротамерів та підтримки планарної структури молекули чи її сегментів (формули 1.3). Нековалентні взаємодії діють між незв’язаними атомами та включають сили Ван–дер–Ваальса (зазвичай описаний потенціалом Леннарда–Джонса) для

врахування поляризації атомів, та кулонівський взаємодії для опису електростатичних взаємодій між зарядженими або частково зарядженими атомами (формули 1.4).

$$U = U_b + U_a + U_d + U_{id} + U_{LJ} + U_c \tag{1.1}$$

$$U_b = \sum \frac{1}{2} k^b{}_{ij} \left(r_{ij} - b_{ij} \right)^2 \qquad U_a = \sum \frac{1}{2} k^\theta{}_{ijk} \left(\theta_{ijk} - \theta_{ijk}{}^0 \right)^2 \tag{1.2}$$

$$U_d = \sum k_\phi \left(1 + \cos\left(n\phi_{ijkl} - \delta_{ijkl} \right) \right) \qquad U_{id} = \sum \frac{1}{2} k_\xi \left(\xi_{ijkl} - \xi_0 \right)^2 \tag{1.3}$$

$$U_{LJ} = \sum_i \sum_{j>i} 4\varepsilon_{ij} \left(\left(\frac{\sigma_{ij}}{r_{ij}} \right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}} \right)^6 \right) \qquad U_c = \sum_i \sum_{j>i} \frac{q_i q_j}{4\pi\varepsilon_0 r_{ij}} \tag{1.4}$$

Розрахунок МД дає ряд значних переваг для дослідження явищ, які відбуваються на молекулярному рівні. Під час розрахунку можна зберігати координати, швидкості та сили, які діють на атоми в будь-який момент часу, що може бути неможливим для звичайного експерименту. Також, умови в яких знаходиться досліджувана система, є точно відомими, і можуть бути ретельно контрольованими. Порівнюючи результати МД, отриманих за різних умов моделювання, можна оцінювати вплив різних чинників на молекулярну систему, наприклад, вплив модифікації послідовності мономерів макромолекули на її конформацію та властивості.

1.2 Практичне використання МД

За останні десятки років відбулось стрімке зростання популярності методів симуляції МД. Однією з причин такого зростання може бути може слугувати, стрімке зростання кількості експериментально визначених структур певних класів макромолекул, які мають вирішальне значення для функціонування живих організмів, такі як білки та нуклеїнові кислоти [2]. Для розрахунку МД, ці експериментально-визначені структури є необхідним

початковим елементом для початку моделювання. За рахунок цього, стало можливим дати відповіді на питанням, які можна вирішити за допомогою МД, наприклад, молекулярні механізми функціонування білків, чому деякі патологічно згорнуті білки агрегують за певних умов, як найкраще здійснити структурний дизайн ліків.

Ще однією важливою причиною зростання популярності є значне покращення ефективності методів розрахунок МД та збільшення їх доступності. В початковий етап розвитку МД для моделювання використовувалось дороговартісне обчислювального обладнання. Значне покращення продуктивності комп’ютерного апаратного забезпечення, зокрема графічних процесорів, дало можливість проводити значущі розрахунки МД на значно ширшому спектрі комп’ютерного обладнання за помірну вартість. Програмні пакети для виконання моделювання МД також стали простішими у використанні, за рахунок розвитку і документації користувацького інтерфейсу. Також наближені фізичні моделі, що лежать в основі моделювання МД, стали значно точнішими [3].

За допомогою розрахунку МД можна дослідити багато явищ, які відбуваються на молекулярному рівні та дати відповідь на велику кількість питань. Особливо важливим застосуванням МД є визначення того, як біомолекулярна система реагує на певні зміни. Наприклад:

1. Змінити молекулярне оточення макромолекули, наприклад, змінивши концентрацію солей в розчині або склад ліпідів у мембрані.
2. Змінити один або більше амінокислотних залишків у амінокислотній послідовності білку, наприклад, щоб пояснити або дослідити функціональний ефект мутації.
3. Замінити молекулу зв’язаного ліганду на іншу. Додати ліганд, якщо його не було, чи повністю прибрати. Це дає змогу дослідити вплив присутності чи відсутності ліганду на конформацію макромолекули.

4. Змінити стан протонування основних чи кислотних амінокислот для дослідження їх ролі в утворенні фінальної структури макромолекули чи взаємодії з лігандом чи іншою макромолекулою.
5. Додавання до амінокислот певних невеликих залишків, наприклад, неорганічного фосфату, для дослідження впливу посттрансляційних модифікації.
6. Застосувати зовнішні сили до обраної групи атомів, наприклад, для дослідження сили взаємодій між двома макромолекулами.

У кожному з цих випадків, як правило, слід виконувати моделювання збуреної та незбуреної систем, щоб визначити стійкі відмінності в результатах.

Моделювання МЛ також можна використовувати для перевірки або вдосконалення точності моделі структури, отриманої експериментальним методами, такими як рентгеноструктурний аналіз, ядерна магнітно-резонансна спектроскопія та кріоелектронна мікроскопія. Отримана такими методами структура може мати змінену конформацію, через перебування в умовах, відмінних від природних. Особливу складність становлять мембранні білки, через неможливість, при фіксації структури, включити подвійний ліпідний шар. Як правило такі проблеми можна вирішити, використавши отриману структуру для моделювання динаміки макромолекули в її нативному середовищі та дозволивши структурі перейти до більш стабільної конформації, якщо така існує.

Ще одним способом застосуванням моделювання МД є оцінка гнучкості або рухливості різних областей макромолекули. Експериментальні методи визначення структури, такі як рентгеноструктурний аналіз та кріоелектронна мікроскопія, зазвичай дають усереднену структуру. Дослідивши динаміку такої структури, можна кількісно визначити, наскільки рухливими є різні ділянки молекули в стані рівноваги та яких типів структурних коливань вони зазнають. Таким чином також можна дослідити

динаміку молекул води та іонів, які часто є критичними для функціонування білка та зв’язування ліганду [4].

Також, дослідження МД можуть бути спрямовані на спостереження за важливими біомолекулярними процесами в дії, такими як індуковані лігандом або напругою конформаційні зміни, зв’язування лігандів чи макромолекул, транспорт речовин через мембрану та згортання білків.

1.3 Програмний пакет GROMACS

GROMACS є одним із найбільш популярних програмних пакетів із відкритим вихідним кодом, який використовується переважно для розрахунку МД біомакромолекул. Він надає велику кількість способів розрахунку динаміки, а також інструментів для її підготовки та аналізу. В ньому також реалізовано кілька передових методів для розрахунків вільної енергії. З виходом версії 5.0 в GROMACS було реалізовано нові та покращені алгоритми розпаралелювання, що дало можливість значно покращити продуктивність розрахунку динаміки. У великій кількості сучасного комп’ютерного обладнання, значну роль в забезпечені високої продуктивності відіграють потужні графічні та центральні процесори. Помітну революцію в області МД, зробили графічні процесори — завдяки ним, висока продуктивність моделювання не тільки стала більш доступною, а й рентабельною. GROMACS підтримує використання графічних процесорів з версії 4.5, яка спочатку була обмежена графічними процесорами Nvidia. З виходом нових версій, підтримку графічних процесорів було розширено та вдосконалено [5]. GROMACS, окрім програмного інтерфейсу CUDA для Nvidia, також підтримує використання OpenCL і SYCL для роботи на графічних процесорах AMD, Apple та Intel, що забезпечує велике прискоренням в розрахунку, порівняно з випадком коли використовуються лише центральні процесори.

МД значно розширює можливості вивчення міжмолекулярних взаємодій, шляхом забезпечення просторової та часової роздільної здатності, недоступної в звичайних експериментах. Результати розрахунків МД стали більш точним за допомогою добре параметризованих силових полів. Для розрахунку МД обирається початкова молекулярна конфігурація, разом з описаними атомними взаємодіями за допомогою силових, після чого запускає розрахунок динаміки разом зі збереженням необхідної інформації. Як правило, таким чином оцінюються велика кількість атомних взаємодій протягом багатьох часових кроків інтегрування, що може вимагати надзвичайної кількості обчислювальної потужності та часу — наукова якість результату часто пропорційна кількості вибірки. Величезний потенціал МД призвів до створення великої кількості програмних пакетів для моделювання включаючи GROMACS, CHARMM [6], NAMD [7], AMBER [8], і OpenMM [9].

Безкоштовна ліцензія, під якою поширюється GROMACS, дозволяє його комерційне використання, внесення модифікації та повторне використання частин або всього коду GROMACS. Інфраструктура GROMACS широко використовується в проектах розподіленого обчислення, таких як Folding@home — створений для розрахунку та дослідження молекулярної динаміки білків, використовуючи волонтерські обчислювальні потужності [10]; EvoGrid — фреймворк, створений для дослідження походження життя [11].

Більшість програмного забезпечення для МД мають високу продуктивність при використанні великої кількості процесорів. Реалізовані методи розрахунку в GROMACS забезпечує найвищу можливу абсолютну продуктивність і ефективність на будь-якому обладнанні, щоб якнайкраще використовувати наявні ресурси. Даний програмний пакет показує високу продуктивність на великій кількості поширених апаратних архітектур, яку мають ноутбуки, звичайні та суперкомп’ютери.

1.4 Реалізовані можливості для моделювання в GROMACS

Всі аспекти розрахунку МД реалізовані в модулі mdrun. В ньому підтримуються інтегрування руху атомів за допомогою методів перекрокування (leap-frog), швидкісного Верле (velocity Verlet), Броунівської та стохастичної динаміки, а також обчислення, які виконують мінімізацію енергії, аналіз нормальних коливань та імітацію відпалу. Підтримуються поширені алгоритми регулювання температури та/або тиску. Для застосування голономних обмежень, щоб усунути високочастотні коливання, і уможливити використання більших часових кроків інтегрування, доступні алгоритми SHAKE [12] та LINCS [13, 14]; останній можна використовувати з віртуальними сайтами взаємодії [15]. В програмний пакет входять поширені силові поля молекулярної механіки CHARMM, AMBER, OPLS та GROMOS у форматі текстових файлів з параметрами, що уможливлює внесення модифікацій та використання інших силових полів зі схожим форматом. Моделювання може використовувати явний або неявний розчинник, застосовувати кілька видів геометричних обмежень, бути атомістичним або грубозернистим. mdrun може одночасно запускати кілька симуляцій, що дозволяє застосовувати методи прискореної вибірки. Доступні можливості застосовувати силу для переміщення певної групи атомів, інструменти для проведення парасолькової вибірки, а також алхімічні методи для розрахунку зміни вільної енергії та основної динаміка [16]. Також доступне здійснення гібридного моделювання квантової механіки/молекулярної механіки (QM/MM), у якому невелика частина системи моделюється за допомогою квантово-механічних обчислень, а решта за допомогою МД. Багато популярних форматів файлів для МД можна читати нативно або через плагін VMD [17].

Окрім вище зазначених можливостей, mdrun також може перераховувати залежних від координат енергетичних величин з готової

траєкторії наданої через опцію -rerun. За допомогою цієї функції можна перерахувати, наприклад, значення ковалентних, нековалентних взаємодій та потенціальну енергію, проте не можна перерахувати енергетичні величини, залежні від швидкостей атомів, такі як кінетична енергія, повна енергія та ентальпія. В даній роботі для mdrun було реалізовано модифіковану версію перерахунку енергій для траєкторії, яка вказується за допомогою нової опції -ererun. Дана модифікована версія для перерахунку енергій, окрім координат, також використовує швидкості для отримання кінетичної енергії та інших залежних від неї величин, що розширює можливості для дослідження динаміки системи. Для модифікованої версії перерахунку також було реалізована можливість зручного вибору окремої групи атомів системи для окремого перерахунку їх енергетичних величин без врахування інших атомів. Це уможливить зручніше дослідження динаміки та можливих змін, які могли відбуватись в межах окремої групи атомів під час МД. Ці групи атомів наприклад, можуть утворювати макромолекули. Можливі групи атомів можуть бути автоматично визначені з топології системи або надані у вигляді індекс файлу (.ndx) через нову опцію -n, яку можна використовувати лише з опцією -ererun.

РОЗДІЛ 2
Методи та результати дослідження

2.1 Початковий код

Для робити, початковий код програмного пакета GROMACS був завантажений з їхнього репозиторію в GitLab де відбувається зберігання та розробка цього програмного забезпечення. Всі поступово внесені зміни зберігалися (фіксувались) за допомогою системи керування версіями git. Збережені зміни знаходяться в окремій гілці, яка має назву «extended-rerun», відгалужена від головної гілки починаючи з фіксації з SHA-1 хешем b4616e9207f610c92e712eda1b7be83a05c0e07c. Усі наведені фрагменти коду, в наступних пунктах, відповідають фіксації з SHA-1 хешем cbc2361ac910892e978c61ed37c03f865f0b1c15. Вихідний код модифікованої версії GROMACS доступний за посиланням (https://github.com/Cyber-Hamster-in-a-jarr/gromacs.git)

Наступні розділи міститимуть опис внесених змін у вихідний код. До опису також будуть наведені фрагменти доданих чи модифікованих рядків коду. З лівого боку цих фрагментів будуть вказані номери рядків за яким вони розташовані у відповідному текстовому файлі. Відсутність номера рядку у фрагменті означатиме, що він утворився внаслідок перенесення і даний рядок є частиною першого верхнього пронумерованого рядка. У фрагментах, чотири крапки «....» позначають код або цілі рядки, які не були змінені чи не є важливими для даного обговорення. Також у верхній частині кожного фрагменту буде вказаний шлях до текстового файлу в якому було внесені зміни. Шлях до текстового файлу вказуватиметься відносно корінної директорії в якій знаходиться код.

2.1.1 Нові опцій для виклику модифікованого перерахунку

Окрім розрахунку МД, mdrun також може читати наявний файл траєкторії і перераховувати лише ті енергетичні величини, які залежать від позиції атомів один відносно одного. Ця функція викликається за допомогою додаткової опції -rerun, яка в якості аргументу приймає назву файлу траєкторії, для якої здійснюватиметься перерахунок. Для того щоб не усувати звичайну версію реалізації перерахунку та для зручності тестування було прийнято рішення створити нову опцію для виклику модифікованої версії. Нова опція -ererun в якості аргументу також приймає назву файлу траєкторії, який окрім координат повинен містити швидкості. Швидкості є необхідними для розрахунку кінетичної енергії та інших залежних від неї величин. Кадри, які не матимуть координат чи швидкостей будуть ігноруватись, тобто для них не відбуватиметься розрахунок енергій.

Для додавання нових опцій до mdrun використовувались наявні об’єкти у початковому коді. На рисунок 2.1 наведений фрагмент коду де до динамічного масиву, з назвою filenames, були додані об’єкти які відповідають новим опціям для mdrun.

src/gromacs/mdrun/legacymdrunoptions.h

```
 99 std::vector<t_filenm> filenames = { { {....},
...                                      ....
111                                     { efTRX, "-ererun", "ererun", ffOPTRD },
112                                     { efNDX, "-n", "index", ffOPTRD },
...                                      ....
129                                     {....} } };
```

Рисунок 2.1. Додавання інформації про нові опції -n та -ererun.

Динамічний масив filenames є членом класу LegacyMdrunOptions і зберігає об’єкти типу struct t_filenm які містять:

1. Ціле число, яке репрезентує тип файлу.
2. Назву опції.
3. Назву файлу за замовченням.

4. Ціле число, яке репрезентує обов’язковість файлу та операції які над ним проводяться (читання/запис).
5. Нову назву файлу, яка була надана опції через командний рядок.

В 111 рядку фрагмента коду на рисунку 2.1 у фігурних дужках вказані значення для об’єкта t_filenm, який зберігатиме інформацію стосовно опції -ererun. Ціле число efTRX вказує на файл типу TRX до якого входять файли з розширенням .xtc, .trr, .cpt, .gro, .g96, .pdb, .tng. "-ererun" і "ererun" вказують на назву опції та назву файлу за замовчуванням відповідно. Ціле число ffOPTRD позначає, що вказаний файл відкривається лише для читання і що дана опція, і відповідно файл, є не обов’язковим в командному рядку.

Нижче, в рядку 112, подібним чином вказана інформація для опції -n. Ціле число efNDX вказує на текстовий файл з розширенням .ndx, який містить додаткові групи індексів атомів системи. Індекс файл також відкривається лише для читання і є не обов’язковим для запуску програми.

Для коректної роботи mdrun, на нові опції, -ererun та -n, були накладені певні обмеження, а саме:

1. Опції -ererun та -rerun не можуть бути одночасно вказані в командному рядку для запуску mdrun.
2. Опцію -n можна застосовувати лише разом з опцією -ererun.

Перше обмеження зумовлене тим, що модифікована версія перерахунку, яка активується опцією -ererun і його початкова версія, активується опцією -rerun, є категорично різними функціями що не можуть працювати одночасно. Друге обмеження зумовлене тим, що опція -n додана для її використання разом з опцією -ererun. Використання -n без -ererun не мала б ніякого функціонального ефекту. Для дотримання цих обмежень було внесено додаткові рядки коду, фрагменти яких наведені на рисунку 2.2.

src/gromacs/mdrun/legacymdrunoptions.cpp

```
158 mdrunOptions.rerun = opt2bSet("-rerun", gmx::ssize(filenames), filenames.data());
159 mdrunOptions.ererun = opt2bSet("-ererun", gmx::ssize(filenames), filenames.data());
160 if (mdrunOptions.rerun && mdrunOptions.ererun)
161 {
162     gmx_fatal(FARGS, "-rerun and -ererun cannot be used at the same time");
163 }
164 if (opt2bSet("-n", gmx::ssize(filenames), filenames.data()) && !mdrunOptions.ererun)
165 {
166     gmx_fatal(FARGS, "Option -n can only be used with -ererun");
167 }
```

Рисунок 2.2. Перевірка дотримання обмежень, накладених на нові опції, -n та -ererun.

В рядках 158 та 159 відбувається перевірка на наявність в командному рядку вказаних опцій -rerun і -ererun разом з їхніми аргументами. Функція opt2bSet в якості аргументів приймає:

1. Назву опції для пошуку.
2. Кількість елементів в масиві.
3. Покажчик до масиву, елементи якого зберігають інформацію про кожну опцію, за допомогою яких вказують назву файлу в командному рядку.

Ця функція здійснює пошук елемента в масиві, який зберігає інформацію для опції з даною назвою, втому числі інформацію про те чи була опція з її аргументом вказана в командному рядку. Функія повертає логічне значення, яке буде істинним (true) якщо опція разом з її аргументом були вказаними в командному рядку, або хибним (false) в іншому випадку. Повернуті значення зберігаються у відповідних змінних rerun і ererun, які є членами структури MdrunOptions.

У випадку коли опції -rerun та -ererun будуть одночасно вказаними в командному рядку, змінні MdrunOptions::rerun та MdrunOptions::ererun одночасно матимуть істинне значення, вираз в операторі if, рядок 160, також матиме істинне значення, що призведе до виклику функції gmx_fatal в рядку 162, яка завершує виконання програми і виводить повідомлення про

виникнення відповідної помилки. Дана конфігурація зумовлює те, що під час виконання лише одна змінна (MdrunOptions::ererun та MdrunOptions::rerun) може мати істинне значення або обидві можуть мати хибне значення.

Оператор if в рядку 164 перевіряє дотримання другої умови. Якщо опція -n була вказана в командному рядку (функція opt2bSet повертає істинне значення) і змінна MdrunOptions::ererun не є істинною (опція -ererun невказана в командному рядку) то загальний вираз в Операторі if рядку 164 є істинним, що вказує на порушення другої умови і, як наслідок, відбувається виклик функції gmx_fatal.

Структура MdrunOptions зберігає інформацію про допоміжні опції, які впливають на роботу та активність певних функцій mdrun. Зокрема початкова версія даної структури містить змінну логічного типу даних з назвою rerun, яка репрезентує активність початкової версії перерахунку (опція -rerun). До цієї структури також була додана логічна зміна для репрезентації активності модифікованого перерахунку (опція -ererun), рисунок 2.3.

src/gromacs/mdtypes/mdrunoptions.h

```
100 struct MdrunOptions
101 {
...     ....
105     gmx_bool ererun = FALSE;
...     ....
132 };
```

Рисунок 2.3. Додавання нової логічної змінної для репрезентації активності модифікованого перерахунку.

2.1.2 Реалізація інтерфейсу для вибору груп атомів

Дана функція забезпечуватиме можливість, перед початком розрахунку, вибрати групи атомів системи для подальшого перерахунку енергій. Можливі групи можуть бути надані у вигляді GROMACS індекс файлу (.ndx) через опцію -n, в якому знаходяться назви груп з індексами

атомів, які належать до них. У випадку відсутності індекс файлу, можливі групи атомів будуть автоматично визначеними виходячи з топології системи. Далі перерахунок енергій здійснюватиметься лише для тих атомів, які знаходяться в обраній групі.

Загалом, послідовність операцій, які реалізовують дану функцію, проходить наступним чином:

1. Через інтерфейс командного рядка, користувач обирає групу атомів з можливих запропонованих, отриманих з індекс файлу або визначених після аналізу топології системи.
2. Збереження індексів атомів обраної групи. Видалення з глобальних координат, швидкостей і топології інформації про інші атоми, зі збереженням інформації про атоми обраної групи.
3. При опрацюванні траєкторії для перерахунку енергій, виділяти координати і швидкості атомів обраної групи за допомогою збережених індексів.

На рисунку 2.4 наведений фрагмент в якому реалізовується отримання даних з файлу запуску (.tpr) та конструювання з них об’єктів, які зберігають початковий мікростан системи, її топологію та параметри для розрахунку МД. В цій частині також відбувається отримання індексів обраної групи атомів.

src/gromacs/mdrun/runner.cpp

```
 897 int Mdrunner::mdrunner()
 898 {
 ...     ....
 913     const bool doERerun = mdrunOptions.ererun;
 ...     ....
 955     std::unique_ptr<t_inputrec> inputrec;
 956     std::unique_ptr<t_state> globalState;
 957     std::unique_ptr<int[]> ERerunIndex;
 958
 959     auto partialDeserializedTpr = std::make_unique<PartialDeserializedTprFile>();
 960
 961     if (isSimulationMainRank)
 962     {
 963         // Allocate objects to be initialized by later function calls.
 964         /* Only the main rank has the global state */
 965         globalState = std::make_unique<t_state>();
 966         inputrec = std::make_unique<t_inputrec>();
 967         int* tempERerunIndex = nullptr;
 968         /* Read (nearly) all data required for the simulation
 969         * and keep the partly serialized tpr contents to send to other ranks later
 970         */
 971         applyGlobalSimulationState(
 972                                    *inputHolder_.get(),
                                    partialDeserializedTpr.get(),
                                    globalState.get(), inputrec.get(), &mtop,
 973                                    tempERerunIndex, doERerun,
 974                        opt2bSet("-n",gmx::ssize(filenames), filenames.data()) ?
                        opt2fn("-n", filenames.size(), filenames.data()) : nullptr);
 975
 976         ERerunIndex.reset(tempERerunIndex);
 ...         ....
 990     }
....     ....
2463 } // Mdrunner::mdrunner
```

Рисунок 2.4. Початок ініціації об’єктів, які зберігають мікростан системи, її топологію та параметри для розрахунку МД.

Клас Mdrunner забезпечує підтримку налаштування та виконання mdrun. Цей клас несе відповідальність за тривалість життя структур даних, які існують протягом життя симуляції, напр. для ведення лог файлу та комунікації між потоками. Сам процес налаштування та запуску МД відбувається у функції mdrunner, яка є членом класу Mdrunner.

В рядку 913 була додана нова логічна змінна doERerun, яка матиме істинне значення, якщо буде здійснюватись модифікований перерахунок енергій. Ця змінна використовується в якості аргументу для деяких функції, а також виразів, для надання інформації про активність модифікованого перерахунку енергій.

В рядках 955 та 956 ініціюються розумні вказівники inputrec і globalState, які посилаються на об’єкти, що зберігають параметри для МД та мікростан системи, відповідно. Ці вказівники отримують адреси цих об’єктів після їх створення в динамічно виділеній пам’яті (рядки 965 та 966). В рядку 957 був доданий новий розумний вказівник ERerunIndex, який посилатиметься на масив з індексами обраної групи атомів або нінащо не посилатиметься (матиме значення nullptr), якщо для перерахунку буде обрана уся система.

В рядку 959 розумний вказівник partialDeserializedTpr посилається на сконструйований в динамічно виділеній пам’яті об’єкт, який зберігатиме прочитані з файлу запуску байти у вигляді динамічного масиву ASCII символів. Інформація, яка зберігається в partialDeserializedTpr, використовується у випадку присутності більше ніж одного потоку чи процесу реалізованих за допомогою бібліотеки thread-MPI або MPI.

Доданий вказівник tempERerunIndex в рядку 967, тимчасово зберігатиме, отриману з функції applyGlobalSimulationState (рядок 971), адресу масиву з індексами обраної групи атомів. Далі адреса з tempERerunIndex буде скопійована в розумний вказівник ERerunIndex (рядок 976). Використання розумних вказівників забезпечить автоматичне знищення об’єктів, на які вони посилаються, і звільнення динамічно виділеної пам’яті після завершення виконання функції Mdrunner::mdrunner.

Наступні модифікації стосуватимуться функції applyGlobalSimulationState (рядок 971) в якій відбувається процес опрацювання файлу запуску, ініціації топології й мікростану системи та отримання індексів обраної групи атомів у випадку активного модифікованого перерахунку. Дана функція приймає наступні аргументи:

1. Клас SimulationInput — об’єкт який зберігає назви файлів запуску та контрольної точки (.cpt).

2. Вказівник до структури PartialDeserializedTprFile — зберігає прочитані з файлу запуску байти у вигляді динамічного масиву ASCII символів, а також дані прочитані в заголовку та інформацію про періодичні умови.
3. Вказівник до структури t_state — зберігає мікростан системи (напр. координати атомів, швидкості та сили які діють на них).
4. Вказівник до структури t_inputrec — зберігає параметри МД, вказані в .mdp файлі.
5. Вказівник до структури gmx_mtop_t — зберігає інформацію про топологію системи.
6. Вказівник до цілого числа, яке використовуватиметься для тимчасово зберігання адреси масиву з індексами обраних атомів.
7. Логічне значення, яке репрезентує активність модифікованого перерахунку.
8. Вказівник на символи, який посилатиметься на масив, в якому зберігатиметься назва індекс файлу, якщо він був вказаний в командному рядку. В іншому випадку, вказівник нінащо не посилатиметься (матиме значення nullptr).

Останні три аргументи були додані в ході внесення модифікацій. Значення останнього аргументу визначається за допомогою умовного оператора. Якщо функція opt2bSet в рядку 974, перед символом «?», повертає істинне значення то буде викликана функція opt2fn, перед «:», яка повертає назву індекс файлу, вказаного в командному рядку. В іншому випадку повертається nullptr і останній аргумент нінащо не посилається.

На рисунку 2.5 наведений фрагмент з повним визначенням функції applyGlobalSimulationState в яку було внесено зміни.

src/gromacs/mdrun/simulationinput.cpp

```
57 void applyGlobalSimulationState(const SimulationInput& simulationInput,
58                                 PartialDeserializedTprFile* partialDeserializedTpr,
59                                 t_state* globalState,
60                                 t_inputrec* inputRecord,
61                                 gmx_mtop_t* molecularTopology,
62                                 int*& ERerunIndex,
63                                 bool doERerun,
64                                 const char* indexFilename)
65 {
66     if (doERerun)
67     {
68         *partialDeserializedTpr =
69             read_tpx_state(simulationInput.tprFilename_, inputRecord,
                               globalState, molecularTopology);
70         molecularTopology->natomsInTopologyFile = molecularTopology->natoms;
71         t_atoms atoms = gmx_mtop_global_atoms(*molecularTopology);
72         int gnx = 0;
73         char* grpname = nullptr;
74         get_index(&atoms, indexFilename, 1, &gnx, &ERerunIndex, &grpname);
75         bool bSel = (gnx != globalState->numAtoms());
76         for (int i = 0; ((i < gnx) && (!bSel)); i++)
77         {
78             bSel = (i != ERerunIndex[i]);
79         }
80         if (bSel)
81         {
82             fprintf(stderr,
83                     "Will recalculate energies for subset %s
                                of original tpx containing %d "
84                     "atoms\n",
85                     grpname,
86                     gnx);
87             reduce_topology_x(gnx, ERerunIndex, molecularTopology,
                                  globalState->x.rvec_array(), globalState->v.rvec_array());
88             globalState->changeNumAtoms(gnx);
89             molecularTopology->moleculeBlockIndices.resize(1);
90             reduce_PartialDeserializedTprFile(partialDeserializedTpr, inputRecord,
                                                  globalState, molecularTopology);
91         }
92         else
93         {
94             sfree(ERerunIndex);
95             ERerunIndex = nullptr;
96         }
97         sfree(grpname);
98         done_atom(&atoms);
99     }
100    else
101    {
102        *partialDeserializedTpr =
103            read_tpx_state(simulationInput.tprFilename_, inputRecord,
                               globalState, molecularTopology);
104    }
105 }
```

Рисунок 2.5. Визначення функції яка опрацьовує файл запуску, ініціює топологію й мікростан системи та отримує індекси обраної групи атомів.

В середині функції за допомогою оператора if-else було створено два блоки. Перший блок, в рядках 67-99, виконується у випадку коли логічна змінна doERerun матиме істинне значення, тобто коли буде проводитись

модифікований перерахунок. В іншому випадку виконуватиметься другий блок в рядках 101-104.

Функція read_tpx_state, в рядках 69 і 103, опрацьовує файл запуску, назва якого вказаний в першому аргументі, та з наявних в цьому файлі даних ініціює мікростан системи, її топологію та параметри для МД, в об’єктах на яких посилаються три вказівники, поданих в якості останніх аргументів функції. Ця функція також створює і повертає структуру PartialDeserializedTprFile, яка копіюється в об’єкт, на який посилається вказівник partialDeserializedTpr.

Функція read_tpx_state опрацьовує всю доступну інформацію з файлу запуску й відповідно мікростан і топологія ініціюються для всієї системи. У випадку активності модифікованого перерахунку (doERerun є істинним), виконуватиметься перший блок (рядки 65-98) де відбувається отримання від користувача обраної групи атомів і подальше зменшення мікростану й топології для зберігання інформації лише про обрану групу атомів. В іншому випадку виконуватиметься другий блок (рядки 101-104), інформація, отримана після роботи функції read_tpx_state, лишається незмінною і функція applyGlobalSimulationState завершує свою роботу.

В першому блоці в рядку 70, в окрему змінну natomsInTopologyFile, яка була додана в структуру gmx_mtop_t (рисунок 2.6), копіюється кількість атомів усієї системи.

api/legacy/include/gromacs/topology/topology.h

```
135 struct gmx_mtop_t
136 {
...     ....
158     int natomsInTopologyFile = 0;
...     ....
197 };
```

Рисунок 2.6. Додавання змінної для зберігання загальної кількості атомів, отриманих з файлу запуску.

В подальшому кількість атомів усієї системи, знадобиться для перевірки відповідності файлу запуску й траєкторії. Оскільки далі при зменшені топології, кількість атомів, яка зберігається для перерахунку, може змінитись, ми зберігаємо їх початкову загальну кількість в окремій змінній.

На рисунку 2.5 в рядку 71, для подальшого аналізу системи на можливі групи атомів, створюється структура t_atoms за допомогою функції gmx_mtop_global_atoms, яка в якості аргументу приймає структуру gmx_mtop_t, з якої отримується інформація про всі атоми системи. Інформація про атоми в структурі t_atoms, включає наприклад їх тип, масу, заряд, належність до певної молекули/залишку та інше. Ці дані зберігаються у вигляді масиві, в динамічно виділеній пам’яті; тому в рядку 98, за допомогою функції done_atom відбувається звільнення виділеної пам’яті для цих масивів.

Функція get_index, в рядку 74, забезпечує аналіз системи на можливу групу атомів та отримання від користувача обраної групи. get_index приймає наступні аргументи:

1. Вказівник на структуру t_atoms яка використовуватиметься для аналізу системи на можливі групи атомів.
2. Вказівник на назву індекс файлу.
3. Ціле число — кількість груп які, користувач може обрати серед можливих. В цьому випадку користувач може обирає одну групу.
4. Вказівник на ціле число, яке зберігатиме кількість атомів у обраній групі.
5. Вказівник, який зберігатиме адресу масиву з індексами обраної групи атомів.
6. Вказівник, який зберігатиме назву обраної групи атомів.

Масив з індексами обраної групи атомів та назва обраної групи атомів розміщуватимуться в динамічно виділеній пам’яті. Вказівник, який зберігає адресу пам’яті зайнятою масивом індексів, контролюється класом

std::unique_ptr, що забезпечує автоматичне звільняє виділеної пам’яті при знищені цього об’єкту (після закінчення МД). Пам’ять, виділена для назви обраної групи, звільняється за допомогою функції free, яка викликається через макрос sfree, в рядку 97.

Після завершення виконання функції get_index, відбувається перевірка на те чи кількість атомів обраної групи (рядок 75) та послідовність їх індексів (рядки 76-79) відрізняється від таких для повної системи. Логічна змінна bSel матиме істинне значення, якщо для перерахунку була обрана частина системи, що спричинить виконання блоку в рядках 81-91, де власне відбувається зменшення мікростану системи та її топології. bSel матиме хибне значення у випадку вибору усієї системи, тоді мікростан і топологія лишатимуться незмінними; динамічно виділена пам’ять, яка в цьому випадку зберігатиме індекси усіх атомів системи, звільняється (рядок 94), а відповідний вказівник прирівнюється до nullptr.

В блоці, де відбувається зменшення мікростану топології (рядки 81-91), функція fprintf (рядки 82-86) виводить в командний рядок повідомлення про назву обраної групи (змінна grpname, рядок 85) та кількість атомів у ній (змінна gnx, рядок 86).

В рядку 87, функція reduce_topology_x забезпечує зменшення топології та мікростану системи, зберігаючи в них інформацію про обрану групу атомів. В якості аргументів приймає кількість атомів у обраній групі, вказівник на масив з індексами обраної групи, вказівник на структуру gmx_mtop_t, яка зберігає топологію системи, вказівники на координати та швидкості атомів системи. Ця функція є частиною модулю convert-tpr де забезпечує функцію створення нового файлу запуску з обраною частиною системи. В цю функцію були внесені незначні зміни, наведені на рисунку 2.7.

В декларації функції reduce_topology_x був прибраний специфікатор static (рядок 236) щоб зробити цю функцію доступною для використання в інших файлах коду програми. Також для запобігання втрати пам’яті, в рядку

262, було додано виклик функції done_atom для звільнення блоків пам’яті на які посилаються вказівники в об’єкті структури t_atoms, перед тим як в рядку 263 цим вказівникам буде присвоєно адреси до інших блоків пам’яті де зберігатиметься інформація про атоми обраної групи.

src/gromacs/tools/convert_tpr.cpp

```
236 void reduce_topology_x(int gnx, int index[], gmx_mtop_t* mtop, rvec x[], rvec v[])
237 {
...     ....
262     done_atom(&mtop->moltype[0].atoms);
263     mtop->moltype[0].atoms = atoms;
...     ....
275 };
```

Рисунок 2.7. Зміни внесені у функції reduce_topology_x.

Далі, в рядку 88 рисунку 2.5, в об’єкті структури t_state, числу, яке зберігає кількість атомів системи, прирівнюється до числа gnx, яке зберігає кількість атомів у обраній групі, через функцію changeNumAtoms, яка є членом даної структури. В рядку 89 функція resize зменшує розмір динамічного масиву moleculeBlockIndices до одного елементу, після чого цей масив зберігає лише перший елемент. Даний масив зберігає елементи структури MoleculeBlockIndices, які містять межі індексів атомів які утворюють подібні молекули. У випадку зменшення системи, вся обрана група атомів в топології зберігається у вигляді одного блоку молекул. Саме тому moleculeBlockIndices масив зменшується до одного елемента, а правильні значення меж індексів будуть встановлені в наступній функції reduce_PartialDeserializedTprFile.

Нова функція reduce_PartialDeserializedTprFile, в рядку 90, була створена для зміни даних в об’єкті структури PartialDeserializedTprFile щоб вони відповідали топології, яка була змінена у функції reduce_topology_x. Це забезпечить отримання правильної інформації іншими MPI або thread-MPI потоками, якщо вони присутні. Функція reduce_PartialDeserializedTprFile виконує операцію зворотну до того, що виконує read_tpx_state — інформацію з наявного мікростану й топології записує як послідовність байтів файлу

запуску в динамічний масив ASCII символів (файл запуску фактично не створюється). Фрагмент визначення функції reduce_PartialDeserializedTprFile наведений на рисунку 2.8. В рядку 3429, за допомогою функції populateTpxHeader, яка в якості аргументів приймає мікростан, параметри для МД та топологію системи, створюється і повертається структура TpxFileHeader. В цій структурі зберігається така інформація, як наявність координат, швидкостей, сил та топології в основній частині (динамічний масив ASCII символів), кількість атомів й розмір основної частини.

src/gromacs/fileio/tpxio.cpp

```
3424 void reduce_PartialDeserializedTprFile(
                                  PartialDeserializedTprFile* partialDeserializedTpr,
3425                              t_inputrec*                 ir,
3426                              t_state*                    state,
3427                              gmx_mtop_t*                 mtop)
3428 {
3429     partialDeserializedTpr->header = populateTpxHeader(*state, ir, mtop);
....     ....
3435     gmx::InMemorySerializer tprBodySerializer(
                                  gmx::EndianSwapBehavior::SwapIfHostIsLittleEndian);
3436
3437     do_tpx_body(&tprBodySerializer,
3438                 &partialDeserializedTpr->header,
3439                 ir,
3440                 mtop);
3441
3442     partialDeserializedTpr->body =
                                 std::move(tprBodySerializer.finishAndGetBuffer());
3443
3444     partialDeserializedTpr->pbcType = ir->pbcType;
3445 }
```

Рисунок 2.8. Визначення функції reduce_PartialDeserializedTprFile.

В рядку 3435, ініціюється об’єкт tprBodySerializer класу InMemorySerializer, який знаходиться в просторі імен gmx. Цей клас містить динамічний масив ASCII символів в який будуть записуватись байти файлу запуску. Функція do_tpx_body, в рядку 3437, проводить запис байтів файлу запуску в динамічний масив об’єкта tprBodySerializer, використовуючи інформацію з об’єктів які зберігають топологію та параметри МД. Далі, в рядку 3442, новий масив з байтами файлу запуску переміщується до об’єкта body, який є членом структури PartialDeserializedTprFile. В рядку 3444 з

параметрів МД копіюється ціле число, яке репрезентує тип періодичних умов, у відповідну зміну в структурі PartialDeserializedTprFile.

2.1.3 Налаштування та запуск mdrun

Для забезпечення правильної конфігурації та запуску модифікованого перерахунку було внесено певні зміни у функцію Mdrunner::mdrunner, а також структури та класи, що стають частиною об’єкта, який безпосередньо підтримує процеси МД, мінімізації енергії чи перерахунку.

Одними з внесених модифікації, у функції Mdrunner::mdrunner, були використання логічної змінної doERerun (рисунок 2.4, рядок 913) у виразах що використовують логічну зміну doRerun (визначена на рядок вище змінної doERerun, на рисунку 2.4 не показано). Дані зміни являють собою заміну використання змінної doRerun на вираз «doRerun || doERerun», який можна подати як «doRerun або doERerun». Зроблено це для того, щоб гілки коду які виконуються при істинному значенні doRerun також виконувались за умови істинного значення doERerun. Після перевірки, наведеній на рисунку 2.2 рядок 160, лише одна із цих змінних може мати істинне значення. Дана заміна була зроблена в наступних рядках функції Mdrunner::mdrunner:

1. 1014 та 1094, у функції gpuAccelerationOfNonbondedIsUseful як останній аргумент. В цій функції фінальне значення виразу використовується для виведення повідомлення в лоґ файл про здійснення усіх розрахунків МД на CPU, у випадку наявності груп для розрахунку їх енергій нековалентних взаємодій та хибного значення виразу «doRerun || doERerun». Функція повертає, чи підтримується застосування GPU для розрахунку нековалентних взаємодії з указаними параметрами.
2. 1128, третій аргумент функції checkUseModularSimulator — повертає, чи використання модульної симуляції є можливим. Дана реалізація

модифікованого перерахунку не підтримує модульну симуляцію, тому при істинності виразу «doRerun || doERerun» ця функція повертатиме хибне значення.

3. 1262, в операторі if, який запускає блок, в якому припиняється виконання програми та виводить повідомлення про помилку, якщо звичайний чи модифікований перерахунок запускається з .tpr файлом де вказано мінімізація енергії.
4. 1458, у функції decideWhetherToUseGpuForUpdate, як передостанній аргумент. Функція повертає, чи є можливим використання GPU для здійснення інтегрування та оновлення мікростану системи. Дана реалізація модифікованого перерахунку не підтримує інтегрування та оновлення на GPU, тому дана функція повертатиме хибне значення при активності звичайного чи модифікованого перерахунку.
5. 1479, у функції decideWhetherToUseGpuForHalo, п’ятий аргумент. Функція повертає, чи є можливим використання GPU для halo exchange. Дана реалізація модифікованого перерахунку не підтримує використання GPU для halo exchange — функція повертатиме хибне значення при активності звичайного чи модифікованого перерахунку.
6. 2276, в умовному операторі, для визначення значення об’єкта класу перечислення GpuApiCallBehavior. GpuApiCallBehavior::Sync якщо є активними звичайний чи модифікований перерахунок та GpuApiCallBehavior::Async у іншому випадку.

Інші зміни, що стосувались конфігурації наведені на рисунку 2.9. В рядку 1432, функція makeUpdateGroups перевіряє умови на можливість використання оновлення груп (update group) та реєструє повідомлення, якщо їх неможливо використати. При можливості застосування оновлення груп, makeUpdateGroups створює і повертає новий об’єкт класу UpdateGroups із усією необхідною інформації. В іншому випадку, виводить повідомлення в лоґ файл про неможливість використання і повертає пустий об’єкт класу

UpdateGroups, який не містить інформацію для оновлення груп. Група оновлення — це групи атомів із залежностями під час оновлення мікростану. Залежностями можуть бути накладені обмеження та віртуальні сайти. Групи оновлення зазвичай можна використовувати, коли обмежені лише зв’язки за участю водню та застосовуються автоматично, коли це можливо. Це покращує продуктивність за рахунок усунення додаткових комунікацій між MPI та OpenMP потоками для застосування обмежень і конструювання віртуальних сайтів.

src/gromacs/mdrun/runner.cpp

```
 897 int Mdrunner::mdrunner()
 898 {
 ...     ....
1417     UpdateGroups updateGroups;
1418     if (....)
1419     {
....         ....
1424     }
1425     else
1426     {
....         ....
1432         updateGroups = makeUpdateGroups(mdlog,
1433                                         std::move(updateGroupingsPerMoleculeType),
1434                                         maxUpdateGroupRadius,
1435                                         useDomainDecomposition,
1436                                         systemHasConstraintsOrVsites(mtop),
1437                                         cutoffMargin,
1438                                         doERerun);
1439     }
....     ....
2287     SimulatorBuilder simulatorBuilder;
....     ....
2311     simulatorBuilder.add(TopologyData(mtop, &localTopology, mdAtoms.get(),
                                           ERerunIndex.get()));
....     ....
2315     // build and run simulator object based on user-input
2316     auto simulator = simulatorBuilder.build(useModularSimulator);
2317     simulator->run();
....     ....
2463 } // Mdrunner::mdrunner
```

Рисунок 2.9. Модифікації внесені до процесу конфігурації та запуску для підтримки модифікованого перерахунку.

У початковій реалізації, при застосуванні алгоритму LINCS разом з активним оновленням груп, відстані між атомами, що піддаються обмеженню, розраховуються без врахувань періодично граничних умов системи. Це є несумісним з перерахунком енергій з наявної траєкторії,

оскільки послідовні мікростани можуть бути розмежовані значним проміжком часу, за який один із пари атомів, що піддаються обмеженню, міг перетнути межі системи і опинитися на її іншому боці, що призведе до значного збільшення відстані та некоректної роботи алгоритму. Тому до функції makeUpdateGroups була додана можливість заборонити застосування оновлення груп у випадку активного модифікованого перерахунку. Внесені зміни наведені на рисунку 2.10.

src/gromacs/mdlib/updategroups.cpp

```
794 UpdateGroups makeUpdateGroups(const gmx::MDLogger& mdlog,
795                 std::vector<RangePartitioning>&& updateGroupingPerMoleculeType,
796                 const real maxUpdateGroupRadius,
797                 const bool useDomainDecomposition,
798                 const bool systemHasConstraintsOrVsites,
799                 const real cutoffMargin,
800                 const bool doERerun)
801 {
...     ....
804     MessageStringCollector messages;
...     ....
811     messages.appendIf(doERerun,
812                       "Update groups not supported by Extended rerun");
...     ....
827     if (!messages.isEmpty())
828     {
829         // Log why we can't use update groups
830         GMX_LOG(mdlog.info).appendText(messages.toString());
831         return UpdateGroups();
832     }
833
834     // Success!
835     return UpdateGroups(std::move(updateGroupingPerMoleculeType),
                                    maxUpdateGroupRadius);
836 }
```

Рисунок 2.10. Модифікації внесені до функції makeUpdateGroups для заборони використання оновлення груп в модифікованому перерахунку.

В рядку 800 до функції було додано новий аргумент doERerun, який позначає активність модифікованого перерахунку. В доданих рядках 811-812, за допомогою функції appendIf, яка є членом класу MessageStringCollector, до об’єкту messages, ініційованого в рядку 804, додається текст повідомлення (другий аргумент) якщо перший аргумент матиме істинне значення. В цьому випадку об’єкт messages матиме хоча б одне повідомлення, якщо doERerun матиме істинне значення. Далі в операторі if (рядок 827), перевіряється

наявність повідомлень в об’єкті messages. Якщо даний об’єкт має хоча б одне повідомлення (не є порожнім) то виконуватиметься блок (рядки 828-832), в якому до лоґ файлу додаватиметься повідомлення про причину/причини неможливості використання оновлення груп (рядок 830) і повертатиметься за замовчуванням ініційований об’єкт класу UpdateGroups, який не міститиме необхідної інформації, що свідчитиме про неможливість використання оновлення груп, після чого функція makeUpdateGroups завершуватиме роботу. В іншому випадку, якщо messages не матиме повідомлень, використання оновлення груп є можливим, тоді функція повертатиме об’єкт класу UpdateGroups, який міститиме інформацію для оновлення груп (рядок 835).

Кінцевим етапом конфігурації (функція Mdrunner::mdrunner рисунок 2.9) є конструювання об’єкту, який безпосередньо виконує роботу, вказану для mdrun (мінімізація енергії, розрахунок МД, перерахунок енергії і т.д.). Цей процес забезпечує об’єкт simulatorBuilder класу SimulatorBuilder (рядок 2287), який за допомогою методу add збирає всю необхідну інформацію для вказаної роботи та конструює об’єкт для її виконання. Об’єкти які виконують вказану роботу для mdrun мають загальну назву — симулятори. Для того щоб передати симулятору вказівник до масиву з індексами обраної групи атомів, було модифіковано клас TopologyData, який конструюється і додається до simulatorBuilder в рядку 2311. внесені зміни до цього класу наведені на рисунку 2.11. В рядку 303, до конструктору класу був доданий аргумент ERerunInd, значення якого копіюватимуться у вказівник ERerunIndex_ (рядок 315) за допомогою списку ініціації в рядку 304, ErerunIndex_(ERerunInd).

src/gromacs/mdrun/simulatorbuilder.h

```
299 class TopologyData
300 {
301     public:
302     //! Build collection from simulation data.
303     TopologyData(const gmx_mtop_t& globalTopology, gmx_localtop_t*
                     localTopology, MDAtoms* mdAtoms, int* ERerunInd) :
304     globalTopology_(globalTopology), localTopology_(localTopology),
                            mdAtoms_(mdAtoms), ERerunIndex_(ERerunInd)
305     {
306     }
...     ....
315     int* ERerunIndex_;
316 };
```

Рисунок 2.11. Модифікації внесені до класу TopologyData для зберігання вказівника до масиву з індексами обраної групи.

Далі, модифікації були внесенні у функцію build, яка є членом класу SimulatorBuilder (рядок 2316). Ця функція з наявної інформації про вказану роботу конструює симулятор, у динамічно виділеній пам’яті, і повертає до неї розумний вказівник. Функція приймає одне логічне значення в якості аргумент, яке вказує на можливість використання модульної симуляції. Зміни внесені в цю функцію наведені на рисунку 2.12.

src/gromacs/mdrun/simulatorbuilder.cpp

```
 68 std::unique_ptr<ISimulator> SimulatorBuilder::build(bool useModularSimulator)
 90 {
...     ....
130     if (useModularSimulator)
131     {
...     ....
133     return std::unique_ptr<ModularSimulator>(new ModularSimulator(
134           std::make_unique<LegacySimulatorData>(....,
...                                                 ....
155                                                 topologyData_->ERerunIndex_,
...                                                 ....
172                                                 simulatorConfig_->mdrunOptions_.rerun),
173                                                 ....)));
174     }
...     ....
176     return std::unique_ptr<LegacySimulator>(new LegacySimulator(....,
...                                           ....
197                                           topologyData_->ERerunIndex_,
...                                           ....
214                                           simulatorConfig_->mdrunOptions_.rerun));
215 }
```

Рисунок 2.12. Модифікації внесені у функцію SimulatorBuilder::build.

У випадку використання модульної симуляції, конструюватиметься симулятор класу ModularSimulator (рядок 133), який зберігатиме розумний вказівник до об’єкту класу LegacySimulatorData (рядок 134) де зберігатиметься дані про вказану роботу. В іншому випадку конструюватиметься симулятор класу LegacySimulator (рядок 176), який успадковує дані та конструктор від класу LegacySimulatorData. Тому при конструюванні LegacySimulator також конструюватиметься частина успадкована від LegacySimulatorData. В обох випадках викликається конструктор LegacySimulatorData, в список аргументів якого були додані вказівник на масив з індексами обраної групи (рядки 155 і 197, отримується з об’єкту topologyData_ класу TopologyData який є членом класу SimulatorBuilder) та логічну змінну яка позначає активність модифікованого перерахунку (рядки 172 і 214, отримується з об’єкту mdrunOptions_ класу MdrunOptions). Відповідні модифікації, внесені в клас LegacySimulatorData наведені на рисунку 2.13.

src/gromacs/mdrun/isimulator.h

```
109 class LegacySimulatorData
110 {
111 public:
112     //! The constructor
113     LegacySimulatorData(....,
...                         ....
134                         int* ERerunInd,
...                         ....
151                         bool doERerun) :
...         ....
173         ERerunIndex_(ERerunInd),
...         ....
190         doERerun_(doERerun)
191     {
192     }
...     ....
237     int* ERerunIndex_;
...     ....
271     bool doERerun_;
272 };
```

Рисунок 2.13. Модифікації внесені до класу LegacySimulatorData для зберігання інформації для модифікованого перерахунку.

2.1.4 Реалізація модифікованого перерахунку

Функція для здійснення модифікованого перерахунку викликатиметься через симулятор класу LegacySimulator. Після створення симулятора, запуск виконання вказаної роботи здійснюється за допомогою функції run (рисунок 2.9, рядок 2317), яка є членом створеного симулятора, в даному випадку — LegacySimulator. У функції run класу LegacySimulator відбувається запуск іншої функції-члена цього класу, яка виконує вказану для mdrun роботу. Модифікації, внесені до функції LegacySimulator::run, наведені на рисунку 2.14. В операторі switch (рядок 54) змінна eI, яка є членом структури t_inputrec, репрезентує метод інтегрування, який був вказаний в .mdp файлі. Від цієї змінної залежить те, які блоки будуть активовані в операторі switch. У випадку коли вказаними інтеграторами є md, bd, sd, md-vv чи md-vv-avek виконуватиметься блок, який охоплює рядки 61-78. В цьому блоці був доданий оператор if (рядки 70-73) в якому, при істинності значення змінної doERerun_, викликається нова функція LegacySimulator::do_extended_rerun (рядок 72) в якій здійснюється модифікований перерахунок. В цьому ж блоці викликається функція do_rerun (рядок 68) для здійснення звичайної версії перерахунку, якщо doRerun_ (рядок 66) є істинним, чи функція do_md (рядок 76) для здійснення розрахунку МД, коли doRerun_ та doERerun_ мають хибне значення.

src/gromacs/mdrun/legacysimulator.cpp

```
 52 void LegacySimulator::run()
 53 {
 54     switch (inputRec_->eI)
 55     {
 56         case IntegrationAlgorithm::MD:
 57         case IntegrationAlgorithm::BD:
 58         case IntegrationAlgorithm::SD1:
 59         case IntegrationAlgorithm::VV:
 60         case IntegrationAlgorithm::VVAK:
 61             if (....)
 62             {
 ..                 ....
 65             }
 66             if (doRerun_)
 67             {
 68                 do_rerun();
 69             }
 70             else if (doERerun_)
 71             {
 72                 do_extended_rerun();
 73             }
 74             else
 75             {
 76                 do_md();
 77             }
 78             break;
...     ....
108     }
109 }
```

Рисунок 2.14. Модифікації внесені до функції LegacySimulator::run для виклику модифікованого перерахунку.

Функція LegacySimulator::do_extended_rerun містить головну обчислювальну петлю в якій відбувається ітеративний розрахунок енергій для послідовних мікростанів, чи їх частин, у файлі траєкторії. Операції, які відбуваються у функції LegacySimulator::do_extended_rerun, під час використання md методу інтегрування, схематично зображено на рисунку 2.15.

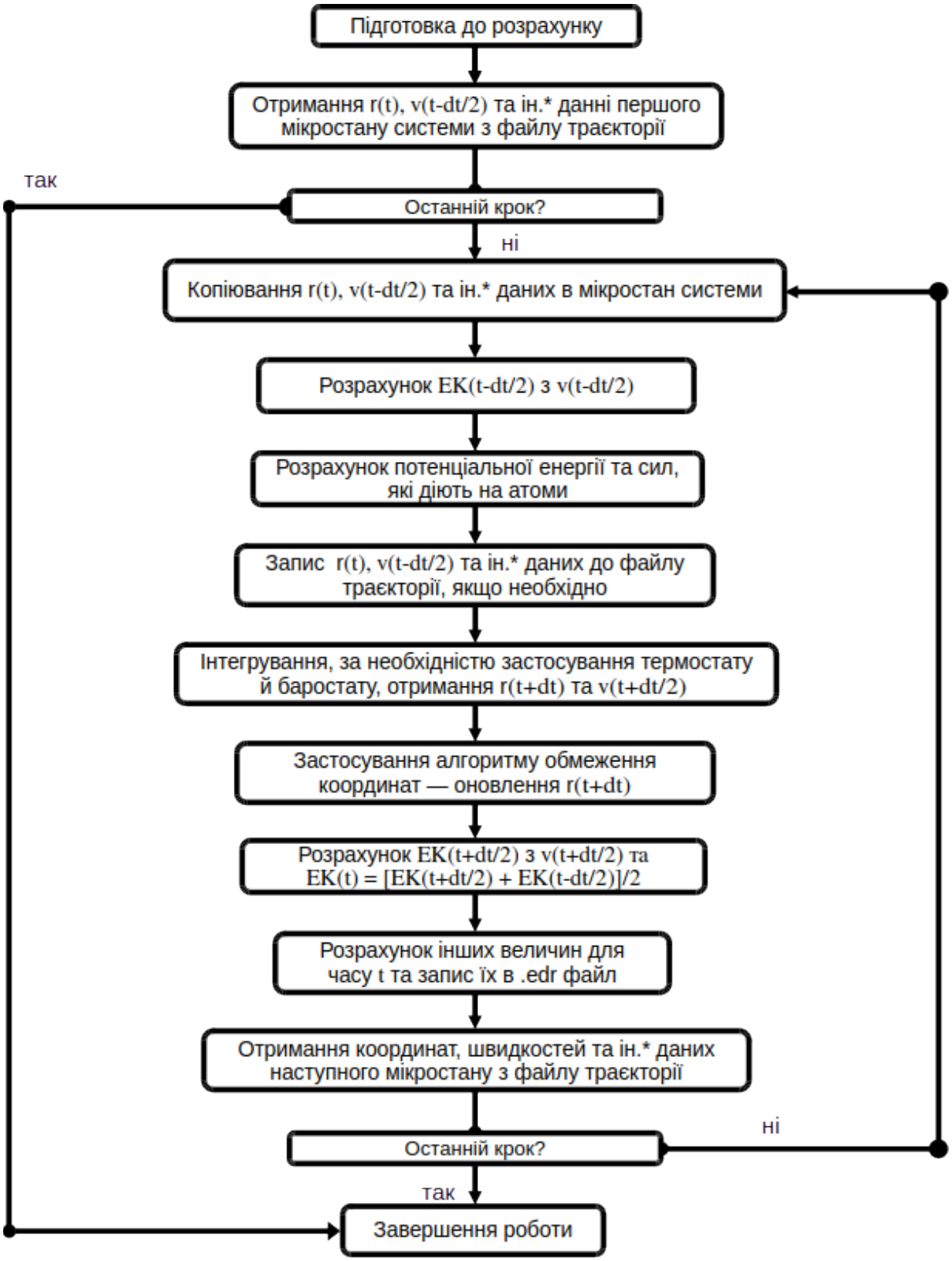

Рисунок 2.15. Блок-схема послідовності головних операцій у функції LegacySimulator::do_extended_rerun у випадку застосування md методу інтегрування. EK — кінетична енергія. * Вектори коробки, номер ітерації, час, кількість атомів.

В mdrun, md інтегратор використовує метод перекрокування (Leapfrog integration) [18] в якому для знаходження координат в час t — r(t) використовується швидкість в час t+dt/2 — v(t+dt/2).

При використанні методу перекрокування, інтегрування здійснюється за формулами 2.1.

$$v(t+dt/2)=v(t-dt/2)+\frac{F(t)}{m}dt$$
$$r(t+dt)=r(t)+v(t+dt/2)dt \qquad (2.1)$$

В початковій реалізації mdrun, кінетична енергія в час t розраховується як середнє кінетичної енергії в час t+dt/2 та t-dt/2 (формули 2.2).

$$EK(t)=\frac{EK(t+dt/2)+EK(t-dt/2)}{2}$$
$$EK(t\pm dt/2)=\frac{1}{2}mv(t\pm dt/2)^2 \qquad (2.2)$$

Оскільки mdrun під час розрахунку МД, використовуючи методу перекрокування, у файл траєкторії, для кожного мікростану, записує r(t) та v(t-dt/2) то в даній реалізації модифікованого перерахунку робиться додатковий крок інтегрування для отримання EK(t) (рисунок 2.13), та інших величин які від неї залежать (повна енергія, ентальпія).

Фрагменти коду, специфічні для роботи модифікованого перерахунку наведені на рисунку 2.16. При підготовці до розрахунку, в блоці оператора if (рядки 917-950), головним потоком/процесом (вираз в операторі if має істинне значення) проводиться відкривання файлу траєкторії та перевірка його відповідності до файлу запуску. В рядку 919, функція read_first_frame відкриває файл траєкторії, вказаний в командному рядку через опцію -ererun (назва файлу повертається функцією opt2fn, третій аргумент), та отримує з нього дані про перший мікростан системи (координати, швидкості, номер ітерації, час, вектор коробки, кількість атомів) які зберігаються в об’єкт rerun_fr структури t_trxframe (четвертий аргумент). Останній аргумент є

цілим числом, яке задає необхідність присутності в кадрах координат й швидкостей — при відсутності одного з них, даний кадр пропускається і функція переходить до наступного.

src/gromacs/mdrun/rerun.cpp

```
 214 void gmx::LegacySimulator::do_extended_rerun()
 215 {
 264     auto setupERerunState = (ERerunIndex_) ? prepareERerunSubState :
                                                  prepareERerunState;
 ...     ....
 917     if (MAIN(cr_))
 918     {
 919         bLastStep = !read_first_frame(oenv_, &status,
                 opt2fn("-rerun", nFile_, fnm_), &rerun_fr,
                 TRX_NEED_X | TRX_NEED_V);
 920         if (rerun_fr.natoms != topGlobal_.natomsInTopologyFile)
 921         {
 922             gmx_fatal(FARGS,
 923                 "Number of atoms in trajectory (%d) does not match the "
 924                 "run input file (%d)\n",
 925                 rerun_fr.natoms,
 926                 topGlobal_.natomsInTopologyFile);
 927         }
 ...         ....
 950     }
....     ....
1030     while (!bLastStep)
1031     {
....         ....
1096         if (MAIN(cr_))
1097         {
....             ....
1107             setupERerunState(rerun_fr, stateGlobal_, constructVsites,
                                  virtualSites_, ERerunIndex_);
1108         }
....         ....
2285         if (MAIN(cr_))
2286         {
2287             /* read next frame from input trajectory */
2288             bLastStep = !read_next_frame(oenv_, status, &rerun_fr);
2289         }
2389     }
2390     /* End of main MD loop */
....     ....
2437 }
```

Рисунок 2.16. Фрагменти коду з функції LegacySimulator::do_extended_rerun в яких відбувається отримання даних про мікростан з файлу траєкторії та їх копіювання в об’єкт який зберігає мікростан системи.

src/gromacs/mdrun/ererun.cpp

```
163 static void prepareERerunSubState(const t_trxframe&          rerunFrame,
164                                    t_state*                  globalState,
165                                    bool                      constructVsites,
166                                    const VirtualSitesHandler* vsite,
167                                    int*                      index)
168 {
169     int curentInd = 0, nAtoms = globalState->numAtoms();
170     auto x = makeArrayRef(globalState->x);
171     auto rerunX = arrayRefFromArray(reinterpret_cast<gmx::Rvec*>(rerunFrame.x),
                                        nAtoms);
172     auto v = makeArrayRef(globalState->v);
173     auto rerunV = arrayRefFromArray(reinterpret_cast<gmx::Rvec*>(rerunFrame.v),
                                        nAtoms);
174     for (int i = 0; i < nAtoms; ++i)
175     {
176         curentInd = index[i];
177         x[i] = rerunX[curentInd];
178         v[i] = rerunV[curentInd];
179     }
180
181     copy_mat(rerunFrame.box, globalState->box);
182
183     if (constructVsites)
184     {
185         GMX_ASSERT(vsite, "Need valid vsite for constructing vsites");

186         vsite->construct(globalState->x, globalState->v, globalState->box,
187                          gmx::VSiteOperation::PositionsAndVelocities);
188     }
189 }
190
191 static void prepareERerunState(const t_trxframe&          rerunFrame,
192                                t_state*                  globalState,
193                                bool                      constructVsites,
194                                const VirtualSitesHandler* vsite,
195                                int*)
196 {
197     auto x = makeArrayRef(globalState->x);
198     auto rerunX = arrayRefFromArray(reinterpret_cast<gmx::Rvec*>(rerunFrame.x),
                                        globalState->numAtoms());
199     auto v = makeArrayRef(globalState->v);
200     auto rerunV = arrayRefFromArray(reinterpret_cast<gmx::Rvec*>(rerunFrame.v),
                                        globalState->numAtoms());
201     std::copy(rerunX.begin(), rerunX.end(), x.begin());
202     std::copy(rerunV.begin(), rerunV.end(), v.begin());
203     copy_mat(rerunFrame.box, globalState->box);
204
205     if (constructVsites)
206     {
207         GMX_ASSERT(vsite, "Need valid vsite for constructing vsites");
208
209         vsite->construct(globalState->x, globalState->v, globalState->box,
                         gmx::VSiteOperation::PositionsAndVelocities);
210     }
211 }
```

Рисунок 2.17. Визначення функцій prepareERerunState та prepareERerunSubState для копіювання даних про мікростан, отриманих з файлу траєкторії.

У випадку успішного отримання усієї необхідної інформації, функція read_first_frame повертає істинне логічне значення. У іншому випадку, функція повертає хибне значення, в результаті змінна bLastStep (рядок 919) матиме істинне значення і головна петля обчислень не виконуватиметься (рядки 1030-2437).

Далі, в операторі if (рядку 920) перевіряється рівність кількості атомів, отриманих з першого мікростану файлу траєкторії та файлу запуску при ініціації мікростану і топології (рисунок 2.5, рядок 70). Нерівність цих значень вказуватиме на можливу невідповідність файлу траєкторії та запуску, що спричинить виклик функції, яка завершує виконання програми та виводить повідомлення про відповідну помилку (рядки 922-926).

Після успішного отримання інформації про перший мікростан (рядок 919) розпочинається виконання головної петлі (рядки 1030-2437) до тих пір поки змінна bLastStep не набуде істинного значення в рядку 2288. Підготовка мікростану до розрахунку відбувається в рядку 1107, де вказівник setupERerunState викликає відповідну функцію, на яку він посилається, для копіювання інформації, отриману з даного кадру файлу траєкторії, в об’єкт, де зберігається мікростан системи. Функція, на яку посилатиметься setupERerunState визначається за допомогою умовного оператора в рядку 264. Якщо вказівник ERerunIndex_ зберігає певну адресу (не дорівнює nullptr), то це означатиме, що було обрану певну частину системи для перерахунку і setupERerunState посилатиметься на функцію prepareERerunSubState. У іншому випадку (ERerunIndex_ = nullptr) setupERerunState посилатиметься на функцію prepareERerunState. Визначення цих функції наведено на рисунку 2.17. обидві функції приймають однаковий набір аргументів:

1. структуру t_trxframe — зберігає інформацію про мікростан, отриману з певного кадру файлу траєкторії.

2. Вказівник до структури t_state — зберігає мікростан системи.
3. Логічне значення, яке вказує на необхідність конструювання віртуальних частинок.
4. Вказівник на клас VirtualSitesHandler — відповідає за конструювання віртуальних чистинок.
5. Вказівник на ціле число — зберігатиме адресу масиву з індексами обраної групи атомів.

Функції prepareERerunSubState та prepareERerunState здійснюють копіювання координат, швидкостей та вектор коробки системи з об’єкту структури t_trxframe в об’єкт структури t_state (рядки 174-181 та 201-203 відповідно) і конструювання віртуальних частинок, якщо вони присутні (рядки 183-188 та 205-210 відповідно). Функція prepareERerunState в мікростан системи копіює всі координати й швидкості (рядки 201-203), отримані з кадру файлу траєкторії (використовується коли для перерахунку була обрана вся система), тоді як функція prepareERerunSubState копіює лише координати та швидкості обраної групи атомів, використовуючи їх індекси в масиві (рядки 174-181).

Інформація про наступний мікростан отримується за допомогою функції read_next_frame (рядок 2288, рисунок 2.16). При успішному отриманні всієї необхідної інформації про мікростан об’єкт rerun_fr міститиме інформацію про новий мікростан, функція повертатиме істинне логічне значення, змінна bLastStep матиме хибне значення, через що буде здійсненна наступна ітерація (повторне виконання рядків 1030-2437). Дана функції також пропускатиме кадри з відсутніми координатами чи швидкостями та переходитиме до наступних. Функція повертатиме хибне значення, коли будуть прочитані всі кадри з файлу траєкторії, змінна bLastStep матиме істинне значення, внаслідок чого відбувається вихід з головної петлі (рядки 1030-2437) та подальше завершення роботи програми.

2.2 Застосування модифікованої версії перерахунку

В якості прикладу, модифікована версія була застосована для виявлення зміни повної енергії окремої молекули РНК в розчині. Система складається з молекули РНК, яка містить сімнадцяти нуклеотидів частина з яких утворюють два шари гуанінового квадруплексу, та водного розчину KCl. Обчислення взаємодії між атомами відбувалося за допомогою силового поля CHARMM [19, 20]. Молекули води описувались за допомогою триточкової моделі TIP3P. Ковалентні зв'язки в яких залучені атоми водню були обмежені за допомогою алгоритму LINCS [14]. Для окремого атома, розрахунок енергії нековалентних взаємодій здійснювався між тими сусідніми атомами, які знаходились на відстані меншій 1.2 нанометра. Для розрахунку дальніх електростатичних взаємодій використано метод підсумовування Евальдової сітки частинок (particle mesh Ewald) [21]. Рівняння руху МД, під час етапу урівноваження і продуктивної динаміки, інтегрувалося за допомогою методу перекрокування (leap-frog algorithm) [18] з кроком в 1 фемтосекунду.

Отримана система піддавалася процесу мінімізації енергії з використання алгоритму найкрутішого спуску з кроком в 1 фемтосекунду. Процес мінімізації відбувався до моменту, коли максимальна сила, яка діє на певний атом в системі, буде меншою ніж 1000 kJ/mol/nm.

Наступний етап урівноваження системи проводились з обмеженням позиції атомів молекули РНК. Перші 125 пікосекунд МД проводилися при температурі 293 K, за умов канонічного ансамблю, з постійною кількістю частинок, об’єму та температури. Контроль температури здійснювався за допомогою термостату Нозе-Гувера [22, 23], баростат не застосовувався. Після урівноваження системи, було запущено розрахунок 100 наносекунд МД з кроком інтегрування в 2 фемтосекунди, за умов ізобарно-ізотермічному ансамблю при температурі 293 K та тиску 1 бар зі застосуванням баростату

Паррінелло-Рахмана [24, 25], термостат також увімкнений. На цьому етапі знімаються обмеження позиції атомів молекули РНК.

В результаті розрахунку МД було отримано файл траєкторії, який містить 1001 кадрів, розмежованих проміжком часу в 0.1 наносекунду. Кожен кадр містить координати атомів системи в час t та їх відповідні швидкості в час t-dt/2. Далі для отриманих позицій та швидкостей атомів молекули РНК було здійснено перерахунок її повної енергії та інших величин, від яких вона залежить (потенціальна та кінетична енергія). Отриманий результат зображений на рисунку 2.18.

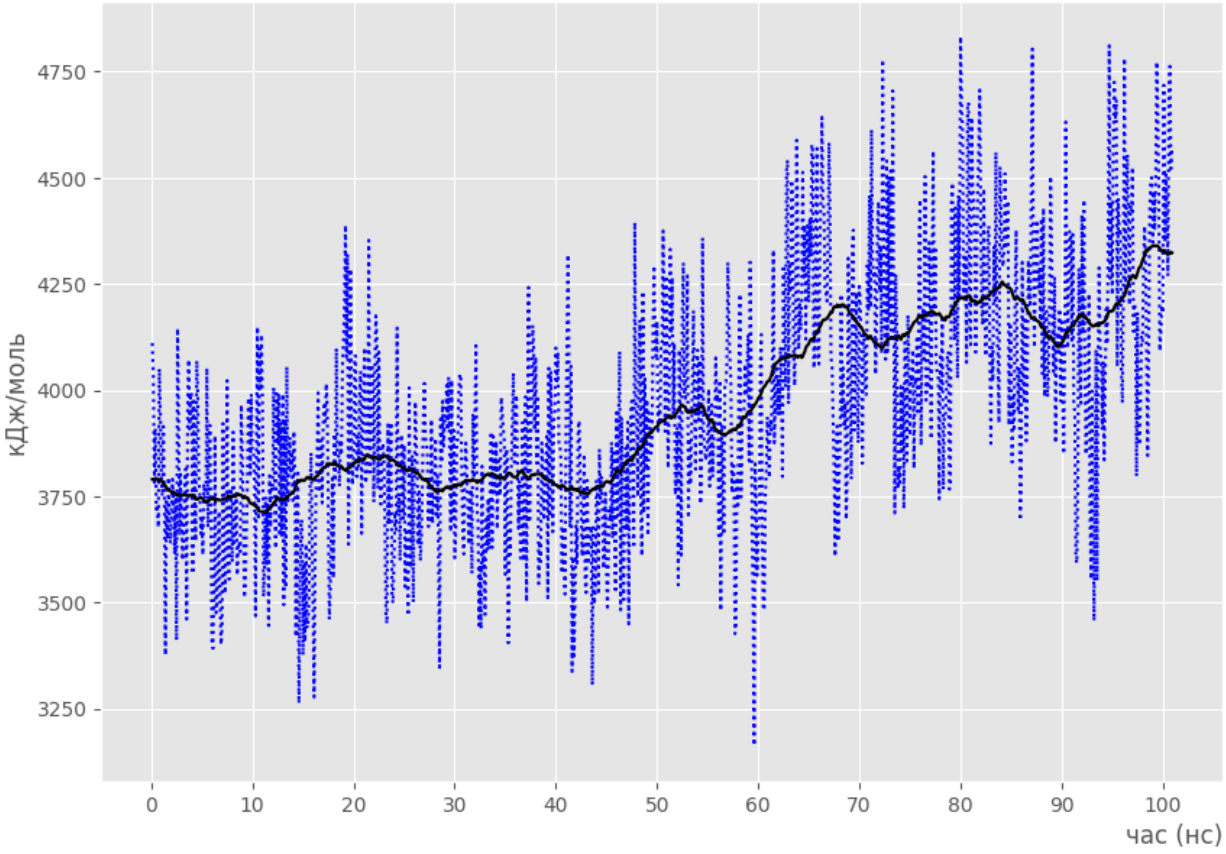

Рисунок 2.18. Повна енергія молекули РНК як функція від часу. Чорною лінією зображено середнє значення.

З графіку на рисунку 2.18 видно що відбувалось поступове збільшення повної енергії. Дане зростання відбувалось головним чином через зростання

потенціальної енергії РНК, оскільки кінетична енергія, за рахунок дії термостату, мала відносно стабільне значення (рисунок 2.19).

Отриманий результат (рисунок 2.18) не враховує нековалентних взаємодій РНК з розчинником, оскільки в цьому випадку перерахунок здійснювався лише для атомів молекули РНК. Якщо розписати повну енергію системи як $V = V_{RNA} + V_{solv} + V_{nb}$ де V_{RNA} — повна енергія молекули РНК; V_{solv} — повна енергія розчинника; V_{nb} — внесок в повну енергію системи нековалентних взаємодій між розчинником і РНК, то V_{nb} можна знайти, віднявши V_{solv} та V_{RNA} від V.

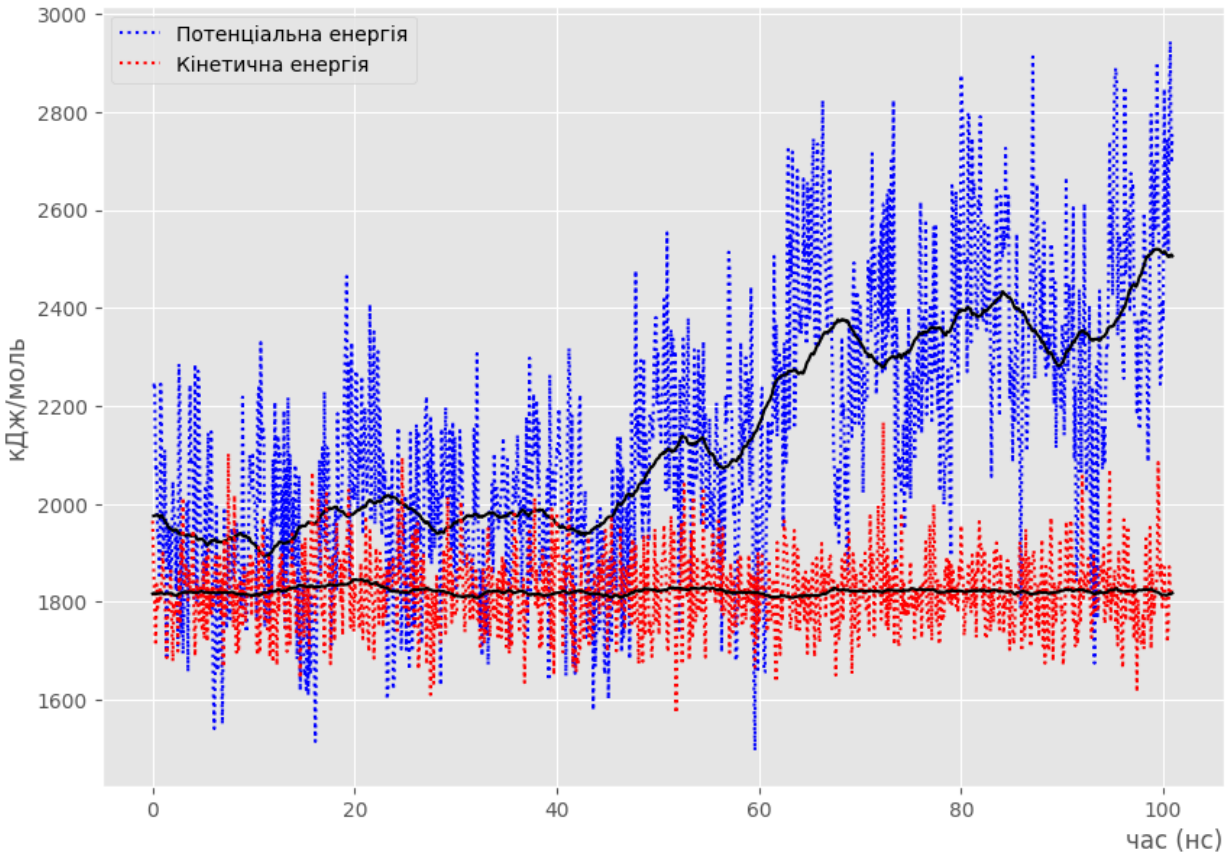

Рисунок 2.19. Потенціальна та кінетична енергія молекули РНК як функція від часу. Чорними лініями зображено їх відповідні середні значення.

Повна енергія системи V є відомою з розрахунку МД а значення V_{solv} та V_{RNA} можна знайти, перерахувавши ці величини окремо для розчинника і

молекули РНК відповідно. При додаванні до повної енергії молекули РНК її нековалентних взаємодій з розчинником можна отримати результат наведений на рисунку 2.20. В цьому випадку можна спостерігати важливу роль, яку відіграють йонні та водневі зв’язки в стабілізації структури молекули РНК.

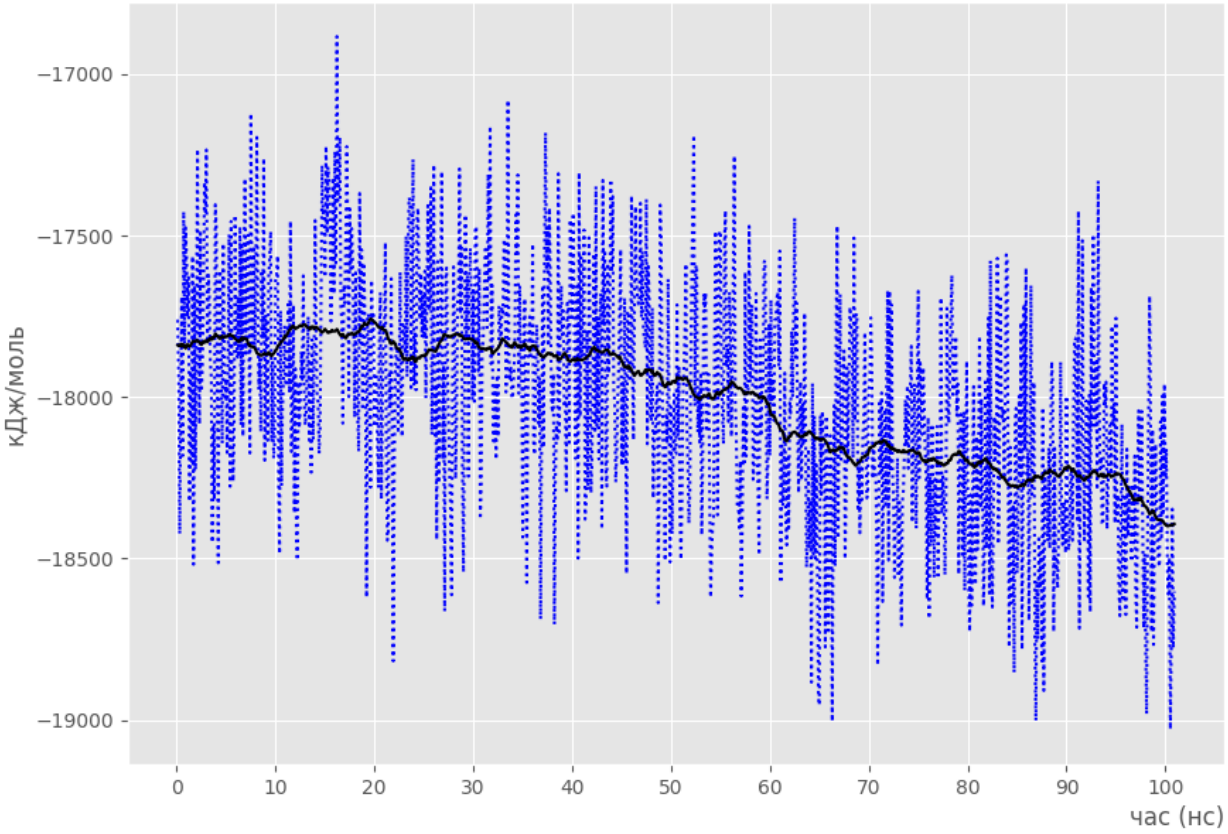

Рисунок 2.20. Повна енергія молекули РНК, з урахуванням її нековалентних взаємодій з розчинником, як функція від часу. Чорною лінією зображено середнє значення.

2.3 Відтворюваність перерахунку кінетичної енергії

Як вже згадувалось раніше — модифікована версія перерахунку використовує швидкості, записані в траєкторію, для розрахунку кінетичної енергії. У випадку використання, для інтегрування руху, методу md (метод перекрокування) кінетична енергія в час t розраховується як середнє

значення кінетичних енергій в час t-dt/2 та t+dt/2 (формула 2.2). Оскільки в кожен кадр траєкторії разом з координатами в час t записуються їх відповідні швидкості в час t-dt/2 то при перерахунку відбувається додатковий крок інтегрування щоб отримання швидкостей в час t+dt/2 для розрахунку кінетичної енергія в час t. Тому, в даній реалізації, при здійсненні перерахунку для окремої групи атомів точне відтворення кінетичної енергії в час t не є можливим, оскільки обрана група атомів знаходитиметься в іншому оточенні (відсутні атоми, які не належать до обраної групи) через що сили, і відповідно швидкості, відрізнятимуться за своєю величиною та напрямком. Для перевірки відтворюваності було здійснено порівняння кінетичної енергії, розрахованої для системи в попередньому розділі, під час МД, з сумою кінетичних енергій, окремо перерахованих для розчинника і молекули РНК. Результат зображено на рисунку 2.21.

З рисунку 2.21 видно, що кінетична енергія, перерахована для окремих компонентів системи є вищою за ту, що вони мали під час МД. середньоквадратична помилка, розрахована для результатів у верхньому графіку рисунку 2.21 становить 137.28 кДж/моль що є меншим за її абсолютні значень порядку 10^5. Також з нижнього графіку рисунку 2.21 видно, що динаміка зміни кінетичної енергії, перерахованої для окремих компонентів системи, співпадає з результати, отриманих з МД. Це робить результат перерахунку кінетичної енергії для окремої групи атомів системи придатним для аналізу динаміки її зміни.

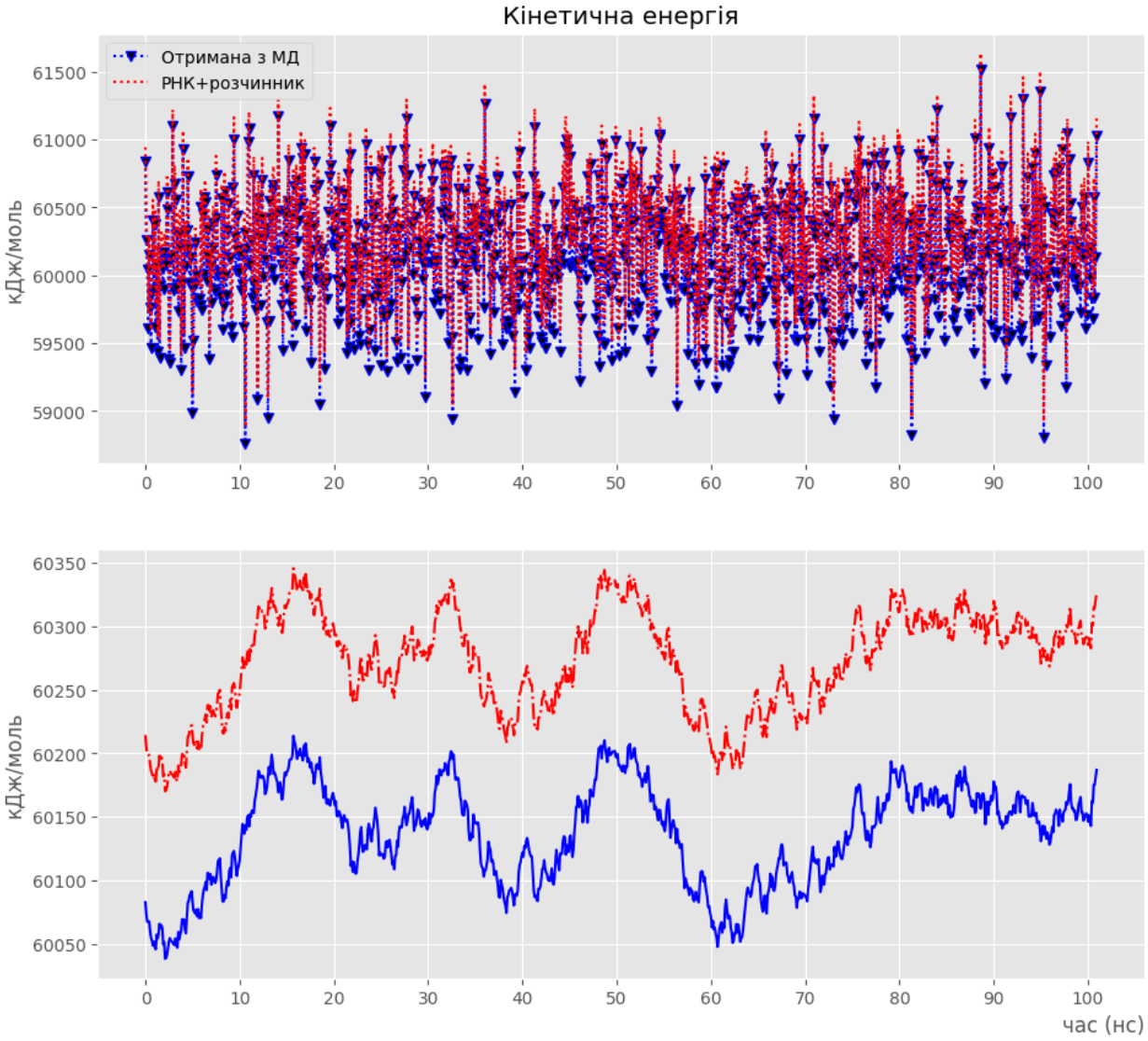

Рисунок 2.21. порівняння кінетичних енергій, отриманої з МД (синій) та сумою окремо перерахованих кінетичних енергій для розчинника та РНК (червоний). Нижче окремо зображено їх усереднені значення.

Потрібно наголосити, що в даній реалізації модифікованого перерахунку, найкраще відтворюються результати для траєкторій, отриманих завдяки інтегруванню руху методом перекрокування (md). У випадку інших методи інтегрування, які підтримує mdrun, наприклад швидкісний метод Верле (velocity Verlet integrator) [26], результати відтворюються значно гірше, через що потрібна подальша робота для усунення цього недоліку.

Висновки

1. В даній роботі було розширено функціонал модуля mdrun, який є частиною програмного пакета GROMAC. Новий функціонал додає можливість перераховувати, з наявної траєкторії, енергетичні величини, які залежать від швидкостей атомів, а також зручніший інтерфейс, який дає можливість, перед початком роботи, обрати окрему частини системи для якої здійснюватиметься перерахунок.
2. Модифіковану версію перерахунку було застосована для виявлення змін в повної енергії окремої молекули РНК, та вплив на цю величину нековалентних взаємодій даної молекули з розчинником, в якому вона знаходиться. Отримані результати підтверджують, що в стабілізації структури молекули РНК, важливу роль відіграють її нековалентні взаємодії з розчинником. Це підтверджує якість обчислювальної моделі, за допомогою якою було отримано дані результати.
3. Важливим наголошенням є рекомендація застосовувати дану версію перерахунку для траєкторії, отриманої шляхом інтегрування руху методом перекрокування (md), оскільки в цьому випадку найкраще відтворюються результати.

Список використаних джерел

1. “GROMACS 2024.1 Source code,” *Zenodo*, Apr. 2024, doi: 10.5281/zenodo.10721181.
2. S. Burley *et al.*, “Protein Data Bank: A Comprehensive Review of 3D Structure Holdings and Worldwide Utilization by Researchers, Educators, and Students,” *Biomolecules*, vol. 12, no. 10, p. 1425, Oct. 2022, doi: 10.3390/biom12101425.
3. K. Lindorff-Larsen, P. Maragakis, S. Piana, M. P. Eastwood, R. O. Dror, and D. E. Shaw, “Systematic Validation of Protein Force Fields against Experimental Data,” *PloS One*, vol. 7, no. 2, p. e32131, Feb. 2012, doi: 10.1371/journal.pone.0032131.
4. S. M. Gopal, F. Klumpers, C. Herrmann, and L. V. Schäfer, “Solvent effects on ligand binding to a serine protease,” *Physical Chemistry Chemical Physics/PCCP*. *Physical Chemistry Chemical Physics*, vol. 19, no. 17, pp. 10753–10766, Jan. 2017, doi: 10.1039/c6cp07899k.
5. S. Páll *et al.*, “Heterogeneous parallelization and acceleration of molecular dynamics simulations in GROMACS,” *Journal of Chemical Physics Online/the Journal of Chemical Physics/Journal of Chemical Physics*, vol. 153, no. 13, Oct. 2020, doi: 10.1063/5.0018516.
6. B. R. Brooks, R. E. Bruccoleri, B. D. Olafson, D. J. States, S. Swaminathan, and M. Karplus, “CHARMM: A program for macromolecular energy, minimization, and dynamics calculations,” *Journal of Computational Chemistry*, vol. 4, no. 2, pp. 187–217, Jun. 1983, doi: 10.1002/jcc.540040211.
7. J. C. Phillips *et al.*, “Scalable molecular dynamics with NAMD,” *Journal of Computational Chemistry*, vol. 26, no. 16, pp. 1781–1802, Oct. 2005, doi: 10.1002/jcc.20289.
8. D. A. Case *et al.*, “The Amber biomolecular simulation programs,” *Journal of Computational Chemistry*, vol. 26, no. 16, pp. 1668–1688, Sep. 2005, doi: 10.1002/jcc.20290.

9. P. Eastman *et al.*, “OpenMM 7: Rapid development of high performance algorithms for molecular dynamics,” *PLOS Computational Biology/PLoS Computational Biology*, vol. 13, no. 7, p. e1005659, Jul. 2017, doi: 10.1371/journal.pcbi.1005659.
10. V. A. Voelz, V. S. Pande, and G. R. Bowman, “Folding@home: Achievements from over 20 years of citizen science herald the exascale era,” *Biophysical Journal*, vol. 122, no. 14, pp. 2852–2863, Jul. 2023, doi: 10.1016/j.bpj.2023.03.028.
11. B. Damer, P. Newman, R. Gordon, T. Barbalet, D. W. Deamer, and R. Norkus, “The EvoGrid: A Framework for Distributed Artificial Chemistry Cameo Simulations Supporting Computational Origins of Life Endeavors,” pp. 73–79, Jan. 2010, [Online]. Available: http://ejournal.narotama.ac.id/files/0262290758chap14.pdf
12. J.-P. Ryckaert, G. Ciccotti, and H. J. C. Berendsen, “Numerical integration of the cartesian equations of motion of a system with constraints: molecular dynamics of n-alkanes,” *Journal of Computational Physics*, vol. 23, no. 3, pp. 327–341, Mar. 1977, doi: 10.1016/0021-9991(77)90098-5.
13. B. Hess, “P-LINCS: A Parallel Linear Constraint Solver for Molecular Simulation,” *Journal of Chemical Theory and Computation*, vol. 4, no. 1, pp. 116–122, Dec. 2007, doi: 10.1021/ct700200b.
14. B. Hess, H. Bekker, H. J. C. Berendsen, and J. G. E. M. Fraaije, “LINCS: A linear constraint solver for molecular simulations,” *Journal of Computational Chemistry*, vol. 18, no. 12, pp. 1463–1472, Sep. 1997, doi: 10.1002/(sici)1096-987x(199709)18:12.
15. H. J. C. Berendsen and W. F. Gunsteren, “Molecular Dynamics Simulations: Techniques and Approaches,” in *Springer eBooks*, 1984, pp. 475–500. doi: 10.1007/978-94-009-6463-1_16.

16. A. Amadei, A. B. M. Linssen, and H. J. C. Berendsen, “Essential dynamics of proteins,” *Proteins*, vol. 17, no. 4, pp. 412–425, Dec. 1993, doi: 10.1002/prot.340170408.

17. W. Humphrey, A. Dalke, and K. Schulten, “VMD: Visual molecular dynamics,” *Journal of Molecular Graphics*, vol. 14, no. 1, pp. 33–38, Feb. 1996, doi: 10.1016/0263-7855(96)00018-5.

18. R. W. Hockney, S. P. Goel, and J. W. Eastwood, “Quiet high-resolution computer models of a plasma,” *Journal of Computational Physics*, vol. 14, no. 2, pp. 148–158, Feb. 1974, doi: 10.1016/0021-9991(74)90010-2.

19. B. R. Brooks *et al.*, “CHARMM: The biomolecular simulation program,” *Journal of Computational Chemistry*, vol. 30, no. 10, pp. 1545–1614, May 2009, doi: 10.1002/jcc.21287.

20. S. Jo, T. Kim, V. G. Iyer, and W. Im, “CHARMM-GUI: A web-based graphical user interface for CHARMM,” *Journal of Computational Chemistry*, vol. 29, no. 11, pp. 1859–1865, Jun. 2008, doi: 10.1002/jcc.20945.

21. T. Darden, D. York, and L. Pedersen, “Particle mesh Ewald: An N·log(N) method for Ewald sums in large systems,” *Journal of Chemical Physics Online/ the Journal of Chemical Physics/Journal of Chemical Physics*, vol. 98, no. 12, pp. 10089–10092, Jun. 1993, doi: 10.1063/1.464397.

22. S. Nosé, “A molecular dynamics method for simulations in the canonical ensemble,” *Molecular Physics*, vol. 52, no. 2, pp. 255–268, Jun. 1984, doi: 10.1080/00268978400101201.

23. W. G. Hoover, “Canonical dynamics: Equilibrium phase-space distributions,” *Physical Review. A, General Physics*, vol. 31, no. 3, pp. 1695–1697, Mar. 1985, doi: 10.1103/physreva.31.1695.

24. M. Parrinello and A. Rahman, “Polymorphic transitions in single crystals: A new molecular dynamics method,” *Journal of Applied Physics*, vol. 52, no. 12, pp. 7182–7190, Dec. 1981, doi: 10.1063/1.328693.

25. S. Nosé and M. L. Klein, “Constant pressure molecular dynamics for molecular systems,” *Molecular Physics*, vol. 50, no. 5, pp. 1055–1076, Dec. 1983, doi: 10.1080/00268978300102851.

26. W. C. Swope, H. C. Andersen, P. H. Berens, and K. R. Wilson, “A computer simulation method for the calculation of equilibrium constants for the formation of physical clusters of molecules: Application to small water clusters,” *Journal of Chemical Physics Online/the Journal of Chemical Physics/Journal of Chemical Physics*, vol. 76, no. 1, pp. 637–649, Jan. 1982, doi: 10.1063/1.442716.