1. Контрольні питання

1. У чому полягають основні способи досягнення паралелізму?

Досягнення паралелізму можливе за умови виконання наступних вимог до архітектурних принципів побудови обчислювального середовища:

- незалежність функціонування окремих пристроїв ЕОМ - ця вимога відноситься всіх основних компонентів обчислювальної системи (пристрої введення-виведення, обробляючим процесорам, пристроям пам'яті);

- надлишковість елементів обчислювальної системи - організація надлишковості може здійснюватися у таких формах: використання спеціалізованих пристроїв (окремі процесори для цілочислової та дійсної арифметики, пристрої багаторівневої пам'яті - регістри, кеш); дублювання пристроїв ЕОМ шляхом використання декількох однотипних обробляючих процесорів чи кількох пристроїв оперативної пам'яті.

Окремою формою забезпечення паралелізму може бути конвеєрна реалізація обробляючих пристроїв, коли виконання операцій в пристроях представляється як виконання послідовності складових операції команд. В результаті при обчисленнях на таких пристроях на різних стадіях обробки можуть знаходитися одночасно декілька різних елементів даних.

2. У чому можуть полягати відмінності паралельних обчислювальних систем?

Паралельні обчислювальні системи - це фізичні комп'ютерні, а також програмні системи, що реалізують той чи інший спосіб паралельну обробку даних на багатьох обчислювальних вузлах.

Типи паралелізму

1. Паралелізм на рівні бітів

Ця форма паралелізму заснована на збільшенні розміру машинного слова. Збільшення розміру машинного слова зменшує кількість операцій, необхідних процесору для виконання дій над змінними, чий розмір перевищує розмір машинного слова. До прикладу: на 8-бітному процесорі потрібно скласти два 16-бітових цілих числа. Для цього спочатку потрібно скласти нижні 8 біт чисел, потім скласти верхні 8 біт і до результату їх складання додати значення прапора переносу. Разом 3 інструкції. З 16-бітовим процесором можна виконати цю операцію однією інструкцією. Історично 4-бітові мікропроцесори були замінені 8-бітними, потім з'явилися 16-бітові та 32-бітові. 32-бітові процесори довгий час були стандартом в повсякденних обчисленнях. З появою технології x86-64 для цих цілей стали використовувати 64-бітні процесори.

2. Паралелізм на рівні інструкцій

Комп'ютерна програма - це, по суті, потік інструкцій, які виконуються процесором. Але можна змінити порядок цих інструкцій, розподілити їх за групами, які будуть виконуватися паралельно, без зміни результату роботи всієї програми. Даний прийом відомий як паралелізм на рівні інструкцій.

Сучасні процесори мають багатоступінчастий конвеєр команд. Кожній ступені конвеєра відповідає певна дія, виконувана процесором в цій інструкції на цьому етапі.

Процесор з N ступенями конвеєра може мати одночасно до N різних інструкцій на різному рівні закінченості. Класичний приклад процесора з конвеєром - це RISC - процесор з 5-ма ступенями: вибірка інструкції з пам'яті (IF), декодування інструкції (ID), виконання інструкції (EX), доступ до пам'яті (MEM), запис результату в регістри (WB).

Деякі процесори, додатково до використання конвеєрів, володіють можливістю виконувати декілька інструкцій одночасно, що дає додатковий паралелізм на рівні інструкцій. Можлива реалізація даного методу за допомогою суперскалярної, коли інструкції можуть бути згруповані разом для паралельного виконання.

3. Паралелізм даних

Основна ідея підходу, заснованого на паралелізмі даних, полягає в тому, що одна операція виконується відразу над всіма елементами масиву даних. Різні фрагменти такого масиву обробляються на векторному процесорі або на різних процесорах паралельної машини. Розподілом даних між процесорами займається програма. Векторизація або розпаралелювання в цьому випадку найчастіше виконується вже на етапі компіляції - перекладу вихідного тексту програми в машинні команди. Роль програміста в цьому випадку зазвичай зводиться до завдання налаштувань векторної або паралельної оптимізації компілятору, директив паралельної компіляції, використанню спеціалізованих мов для паралельних обчислень.

4. Паралелізм завдань

Стиль програмування, заснований на паралелізмі завдань, має на увазі, що обчислювальна задача розбивається на декілька відносно самостійних підзадач і кожен процесор завантажується своєї власної підзадачі.

3. Що покладено в основу класифікації Флінна?

В її основу покладено поняття потоку, під яким розуміється послідовність елементів, команд або даних, що обробляються процесором. Залежно від кількості потоків команд і потоків даних Флінн виділяє чотири класи архітектур:

1. SISD (Single Instruction Stream / Single Data Stream) - одиночний потік команд і одиночний потік даних.
2. MISD (Multiple Instruction Stream / Single Data Stream) - множинний потік команд і одиночний потік даних.
3. SIMD (Single Instruction Stream / Multiple Data Stream) - одиночний потік команд і множинний потік даних.
4. MIMD (Multiple Instruction Stream / Multiple Data Stream) - множинний потік команд і множинний потік даних.

4. У чому полягає принцип поділу багатопроцесорних систем на мультипроцесори і мультикомп’ютери?

В залежності від організації підсистем оперативної пам'яті багатопроцесорні системи можна розділити на два наступних класи.

Системи із спільною (такою, що розділяється) пам'яттю — **мультипроцесори**, у яких є одна віртуальна пам'ять, а всі процесори мають однаковий доступ до даних та командам, що зберігаються в цій пам'яті . За цим принципом побудовані векторні паралельні процесори та симетричні мультипроцесори .

Системи з розподіленою пам'яттю - **мультикомп'ютери**, у яких кожен процесор має

свою локальну оперативну пам'ять, а в інших процесорів доступ до цієї пам'яті відсутній.

5. Які класи систем відомі для мультипроцесорів?

Симетричні мультипроцесори (SMP): У цій системі кожен процесор має однаковий доступ до пам'яті та взаємодіє з іншими процесорами. Система SMP дозволяє будь-якому процесору виконувати будь-яку роботу та ділити ресурси.

6. У чому полягають позитивні і негативні сторони симетричних мультипроцесорів?

Позитивні сторони SMP систем:

1. **Легкість програмування**: Програми для SMP систем можна розробляти без особливих зусиль для управління розподіленими ресурсами, оскільки процесори мають спільний доступ до пам'яті та взаємодіють за допомогою спільних шин або інших засобів комунікації.
2. **Підвищена продуктивність для багатьох завдань**: SMP системи можуть ефективно виконувати паралельні завдання, що дозволяє розподіляти обчислювальні навантаження між процесорами та прискорювати виконання завдань.
3. **Скалійованість**: В SMP системах можна додавати нові процесори для підвищення продуктивності без значних змін у програмному забезпеченні, що робить їх скальованими.

Негативні сторони SMP систем:

1. **Вартість**: SMP системи можуть бути великою фінансовою витратою, оскільки вони вимагають великої кількості процесорів, спільного доступу до швидкої пам'яті та інфраструктури для комунікації між процесорами.
2. **Складність управління**: Управління пам'яттю та розподілом завдань між процесорами може бути складним завданням. Це може призвести до виникнення конфліктів та інших проблем у взаємодії між процесорами.
3. **Обмежена масштабованість**: Деякі SMP системи можуть мати обмеження на кількість підключених процесорів, що обмежує їхню масштабованість у порівнянні з іншими архітектурами, такими як кластери або грід-системи.
4. **Спільний доступ до ресурсів**: Спільний доступ до пам'яті та інших ресурсів може стати джерелом конфліктів, якщо не дотримуватися правильного управління ресурсами.

7. Які класи систем відомі для мультикомп’ютерів?

Кластери (Cluster Computing): Кластер - це мережа незалежних комп'ютерів, які спільно вирішують завдання. Кластери можуть бути використані для розподілених обчислень, обробки великих даних і симуляцій.

8. У чому полягають позитивні і негативні сторони кластерних систем?

Позитивні сторони кластерних систем:

1. **Висока продуктивність**: Кластери можуть забезпечувати високу продуктивність завдяки паралельним обчисленням на кількох вузлах, що дозволяє розподіляти завдання та прискорювати обчислення.
2. **Скалійованість**: Додавання нових вузлів до кластера може покращити продуктивність без необхідності значних змін у програмному забезпеченні, що робить їх скальованими.
3. **Висока доступність**: Кластери можуть бути налаштовані для резервного копіювання та відновлення, що дозволяє забезпечити високу доступність та надійність системи.
4. **Гнучкість та конфігуруємість**: Кластерні системи можуть бути легко налаштовані та розширені відповідно до потреб користувачів, забезпечуючи гнучкість у виборі обладнання та програмного забезпечення.

Негативні сторони кластерних систем:

1. **Складність налаштування та управління**: Кластери вимагають спеціальних знань для налаштування, управління та відладки, що може бути складним завданням, особливо для великих кластерів.
2. **Вартість**: Побудова та підтримка кластерної системи може бути дорогою, оскільки вимагає закупівлі великої кількості обладнання та інфраструктури.
3. **Специфічність задач**: Не всі завдання можуть бути ефективно розпаралельлені для використання в кластерах, що може призвести до недооптимізованого використання ресурсів.
4. **Мережева завадозахищеність**: Збільшення кількості взаємодіючих вузлів у кластері може збільшити ризик мережевих атак та збоїв у роботі системи.

9. Які топології мереж передачі даних найбільш широко використовуються при побудові багатопроцесорних систем?

1) Повний граф

2) Лінійка (Ланцюг)

3) Кільце

4) Зірка

5) Решітка (двовимірна чи тривимірна)

10. У чому полягають особливості мереж передачі даних для кластерів?

Мережа кластера в першу чергу призначена не для зв'язку машин, а для зв'язку обчислювальних процесів. Тому чим вищою буде пропускна здатність мережі, тим швидше будуть вважатися паралельні завдання, запущені на кластері.

11. Які основні характеристики мереж передачі даних?

1. **Пропускна здатність (Bandwidth)**: Це максимальний обсяг даних, який може бути переданий через мережу протягом певного часу. Вимірюється у бітах за секунду (bps), кілобітах за секунду (Kbps), мегабітах за секунду (Mbps) або гігабітах за секунду (Gbps).
2. **Затримка (Latency)**: Це час, який потрібно для передачі сигналу від відправника до отримувача. Затримка може виникати через різні фактори, такі як обробка в мережевих пристроях, час передачі по мережі тощо.
3. **Пропагаційна затримка (Propagation Delay)**: Це час, який потрібний сигналу, щоб пройти від відправника до отримувача через фізичний канал зв'язку.
4. **Джиттер (Jitter)**: Нестабільність у часі прийому даних. Джиттер може виникати через зміни у пропускній здатності мережі або затримки в мережевих пристроях.
5. **Надійність (Reliability)**: Міра того, наскільки надійно дані можуть бути передані та отримані в мережі без помилок чи втрат.
6. **Топологія (Topology)**: Це структура мережі, яка визначає, як вузли та пристрої з'єднані між собою. Популярні топології включають зірку, лінійку, кільце, решітку тощо.
7. **Протоколи передачі даних (Data Transmission Protocols)**: Стандарти та правила, які визначають, як дані передаються в мережі. Приклади включають TCP/IP, Ethernet, Wi-Fi тощо.
8. **Масштабованість (Scalability)**: Здатність мережі збільшувати розмір та пропускну здатність для відповіді на збільшені потреби користувачів чи пристроїв.
9. **Безпека (Security)**: Заходи та протоколи, які вживаються для захисту даних від несанкціонованого доступу, зміни чи втрати.
10. **Вартість (Cost)**: Загальні витрати на розгортання, управління та підтримку мережі.

12. Які системні платформи можуть бути використані для побудови кластерів?

1. **Спеціалізовані кластерні сервери**: Це сервери, які спеціально розроблені та виготовлені для використання в кластерних конфігураціях. Вони можуть бути оптимізовані для паралельних обчислень та мають високу пропускну здатність мережі.
2. **Загальнопризначені сервери**: Звичайні сервери, які використовуються в інфраструктурі підприємства, також можуть бути використані для побудови кластерів. Їх можна об'єднати в кластер, встановивши спеціальне програмне забезпечення.
3. **Високопродуктивні обчислювальні вузли (HPC Nodes)**: Це вузли, спроектовані спеціально для високопродуктивних обчислень. Вони зазвичай мають потужні процесори, велику кількість оперативної пам'яті та можуть бути використані для складних обчислень та наукових досліджень.
4. **GPU-прискорювачі (Graphics Processing Units)**: Використання GPU у кластерах дозволяє розпаралелювати обчислення та прискорити виконання деяких завдань, таких як глибоке навчання та обчислення штучного інтелекту.
5. **Хмарні ресурси**: Хмарні послуги також можуть бути використані для створення кластерів. Публічні хмарні платформи, такі як Amazon Web Services (AWS), Microsoft Azure та Google Cloud Platform (GCP), надають можливість створення кластерів на основі віртуальних машин.
6. **Вбудовані системи та Інтернет речей (IoT)**: Для деяких застосувань можна використовувати вбудовані системи та пристрої Інтернету речей як вузли кластера, забезпечуючи розподілені обчислення та збір даних.

2. Контрольні питання

1. Як визначається модель "операція - операнди"?

у вигляді ациклічного орієнтованого графа

2. Як визначається розклад для розподілу обчислень між процесорами?

1. **Статичний Розклад (Static Scheduling)**: У статичному розкладі завдання розподіляються між процесорами на основі заздалегідь відомої інформації про завдання та ресурси. Цей розклад встановлюється до початку виконання програми та залишається незмінним протягом всього виконання завдання.
2. **Динамічний Розклад (Dynamic Scheduling)**: У динамічному розкладі рішення про розподіл завдань приймаються під час виконання програми, на основі поточних умов та навантаження системи. Динамічний розклад може адаптуватися до змінюючихся умов та розподіляти завдання динамічно між процесорами.
3. **Оперативний Розклад (Task Scheduling)**: Це планування виконання конкретних задач чи задачевих частин між процесорами в межах багатопроцесорної системи. У цьому контексті задачі можуть бути розподілені на різні процесори в залежності від їхньої готовності та пріоритету.
4. **Контекстно-Залежний Розклад (Context-Aware Scheduling)**: У цьому підході розклад враховує контекстні фактори, такі як стан системи, обсяг завдань та навантаження на мережу.
5. **Децентралізований Розклад (Decentralized Scheduling)**: У системах, де кілька процесорів може приймати рішення про розподіл завдань самостійно, застосовується децентралізований розклад.

3. Як визначається час виконання паралельного алгоритму?

Час виконання
Прискореннння = час послідовного/час паралельного
Ефективність = прискорення / кількість використаних ресурвіс або потоків

4. Який розклад є оптимальним?

1. **Статичний Розклад (Static Scheduling)**:
   - **Переваги**: Простота у реалізації, може бути оптимальний для деяких типів завдань, дозволяє використовувати оптимізації на етапі компіляції.
   - **Недоліки**: Не ефективний для змінюючихся навантажень чи розподілу ресурсів, не враховує динамічних змін у системі.
2. **Динамічний Розклад (Dynamic Scheduling)**:
   - **Переваги**: Адаптивний до змінюючихся умов та навантажень, може використовуватися для реакції на непередбачені події чи зміни в середовищі виконання.
   - **Недоліки**: Може виникати велика накладна згідності через динамічне прийняття рішень, може важко досягти оптимальності через нестабільність розподілу завдань.
3. **Оперативний Розклад (Task Scheduling)**:
   - **Переваги**: Дозволяє вибірково розподіляти конкретні завдання на вузли системи, ефективний для задач, які потребують різної кількості ресурсів.
   - **Недоліки**: Складний у реалізації, може потребувати додаткового управління та комунікації між вузлами.
4. **Контекстно-Залежний Розклад (Context-Aware Scheduling)**:
   - **Переваги**: Адаптивний до контексту та змінюючихся умов, може оптимізувати розподіл завдань на основі реальних даних про систему.
   - **Недоліки**: Складніше у реалізації через потребу відслідковування та аналіз змінюючихся умов.
5. **Децентралізований Розклад (Decentralized Scheduling)**:

- **Переваги**: Покращує масштабованість та резистентність до відмов, може бути більш ефективний в умовах великої динамічності.
- **Недоліки**: Складніше у реалізації, може виникати більше проблем з управлінням та синхронізацією.

5. Як визначити мінімально можливий час вирішення завдання?

Мінімально можливий час виконання паралельного алгоритму визначається довжиною максимального шляху обчислювальної схеми алгоритму

6. Що розуміється під паракомп'ютером і для чого може виявитися корисним дане поняття?

система з необмеженою кількістю процесорів

7. Які оцінки слід використовувати в якості характеристики часу послідовного вирішення завдання?

Оцінка О – велике. Використання памяті.

8. Як визначити мінімально можливий час паралельного рішення задачі по графу "операнди - операції"?

Мінімальний час паралельного вирішення задачі за допомогою графу "операнди - операції" може бути визначений за допомогою концепції критичного шляху (critical path). Критичний шлях - це найкоротший шлях у графі від вхідних операцій до вихідних операцій, який визначає мінімальний час, який може бути досягнутий для виконання задачі.

9. Які залежності можуть бути отримані для часу паралельного рішення задачі при збільшенні або зменшення кількості використовуваних процесорів?

1. **Лінійна Залежність (Linear Scaling)**:
   - З деякими завданнями може спостерігатися пряма лінійна залежність: чим більше процесорів, тим швидше виконується завдання. Це може бути характерно для завдань, які можуть бути розділені на незалежні підзавдання та виконувати їх паралельно без необхідності взаємодії між ними.
2. **Сублінійна Залежність (Sublinear Scaling)**:
   - При збільшенні кількості процесорів виходить насичення (saturation point), і додаткові процесори призводять до меншого приросту продуктивності. Це може відбуватися через накладні витрати на комунікацію, синхронізацію або інші обмеження.
3. **Закон Амдала (Amdahl's Law)**:
   - Закон Амдала вказує на те, що час виконання задачі з паралельною частиною може бути обмежений найпомільшою частиною задачі, яка не може бути паралельно вирішена. Зі збільшенням кількості процесорів, які можуть брати участь у паралельній частині задачі, загальний час виконання може зменшитися, але завжди буде існувати обмеження часу виконання внаслідок послідовної частини задачі.
4. **Збільшення Затримок (Increasing Overheads)**:
   - При збільшенні кількості процесорів може збільшуватися накладна затримка на комунікацію та синхронізацію між процесорами, що призводить до зростання загального часу виконання.
5. **Зниження Затримок (Reducing Overheads)**:
   - Якщо вдається зменшити накладні витрати на комунікацію та синхронізацію (наприклад, за допомогою оптимізацій алгоритмів), можна досягти кращої масштабованості та скорочення часу виконання задачі при збільшенні кількості процесорів.

10. При якому числі процесорів можуть бути отримані часи виконання паралельного алгоритму, порівнянні по порядку з оцінками мінімально можливого часу рішення задачі?

Зазвичай при збільшенні кількості процесорів спочаткуспостерігається покращення продуктивності, проте після певної точки зростання кількості процесорів накладні витрати можуть призводити до того, що збільшення клікості процесорів не призводить до подальшого зменшення часу виконання задачі. Ця точка настає, коли співвідношення між часом виконання паралельної частини та накладними витратами на комунікацію та синхронізацію призводить до того, що збільшення кількості процесорів не приносить значущого приросту у продуктивності.

11. Як визначаються поняття прискорення і ефективності?

Прискорення = час послідовного/час паралельного

Ефективність = прискорення / кількість використаних ресурвіс або потоків

12. Чи можливе досягнення надлінійного прискорення?

Надлінійне прискорення (superlinear speedup) - це ситуація, коли час виконання задачі на паралельній системі з кількома процесорами є меншим за час виконання цієї ж задачі на послідовній системі. Тобто, якщо ви додаєте більше ресурсів (процесорів) до задачі, її виконання стає не просто швидше, а надзвичайно швидше, що може здатися дивовижним.

13. У чому полягає суперечливість показників прискорення і ефективності?

Суперечливість виникає тоді, коли прискорення може збільшуватися при збільшенні кількості процесорів, але ефективність може зменшуватися. Іншими словами, хоча задача може бути вирішена швидше при використанні більшої кількості процесорів, ефективність (тобто ефективне використання цих процесорів) може зменшуватися через накладні витрати на комунікацію та синхронізацію між процесорами.

14. Як визначається поняття вартості обчислень?

1. **Часовий Аспект**: Вартість обчислень може вимірюватися в часі, який потрібен для виконання певних обчислень. Це може бути критичним для додатків, які вимагають великої швидкості виконання, наприклад, у фінансах, медицині або великих наукових дослідженнях.
2. **Ресурсний Аспект**: Вартість обчислень може бути пов'язана з використанням обчислювальних ресурсів, таких як процесорний час, оперативна пам'ять, мережеві ресурси тощо.
3. **Споживання Енергії**: У зв'язку з ростом уваги до екологічних питань, вартість обчислень може бути оцінена через споживання електроенергії та його вплив на навколишнє середовище.
4. **Якість та Точність**: У деяких випадках, особливо в наукових та інженерних дослідженнях, вартість обчислень може бути пов'язана з якістю та точністю результатів. Це може включати в себе вартість витрачених ресурсів для виправлення помилок та підвищення надійності обчислень.

15. У чому полягає поняття вартісно-оптимального алгоритму?

Поняття вартісно-оптимального алгоритму відноситься до вибору алгоритму для вирішення конкретної задачі, який має оптимальне співвідношення між вартістю його виконання та якістю (чи ефективністю) отриманих результатів. Це може бути важливим в багатьох областях, включаючи комп'ютерні науки, бізнес-аналітику, наукові дослідження та інші сфери, де потрібно знайти оптимальний баланс між витратами ресурсів і якістю отриманих даних чи результатів.

16. У чому полягає проблема розпаралелювання послідовного алгоритму підсумовування числових значень?

1. **Залежності Даних (Data Dependencies)**: Підсумовування числових значень зазвичай вимагає послідовності операцій, оскільки кожне наступне значення має бути додане до попереднього сумарного результату. Це створює залежності даних між ітераціями циклу або елементами масиву, що ускладнює паралельне виконання.

2. **Конфлікти Доступу (Access Conflicts)**: Коли багато процесорів намагаються отримати доступ до спільної пам'яті для зчитування та запису, можуть виникати конфлікти доступу, які призводять до очікування та накладають затримки.
3. **Накладні Витрати на Синхронізацію (Synchronization Overheads)**: Під час паралельного виконання даних є необхідність управління синхронізацією, щоб забезпечити коректність підсумовування. Це може призводити до витрат часу на синхронізацію потоків.
4. **Розбиття Даних (Data Partitioning)**: Для ефективного паралельного підсумовування даних може бути необхідно вдало розподілити дані між процесорами. Це може бути складно здійснити, особливо якщо обсяг даних невеликий або не рівномірний.
5. **Геометрія Даних (Data Granularity)**: Розпаралелювання великої кількості дрібних операцій може призвести до надмірної накладної витрати на управління потоками та синхронізацію, що може перевищити вигоди від паралельного виконання.

17. У чому полягає каскадна схема підсумовування? З якою метою розглядається модифікований варіант даної схеми?

Каскадна схема підсумовування є методом обчислення суми числових значень, який використовує послідовний підхід до додавання чисел. В цій схемі сума обчислюється послідовно: перше число додається до другого, результат додається до третього, і так далі, поки не будуть оброблені всі числа. Ця схема є простою та інтуїтивно зрозумілою, але не є оптимальною з точки зору паралельного обчислення через велику кількість залежностей даних та невеликий обсяг паралельних операцій.

Модифікована схема – всі просумовувані значення поділяються на (n/log2n) груп, у кожній з яких містяться log2n елементів. Для кожної групи обчислюється сума значення за допомогою послідовного алгоритму сумовування. На другому етапі для отриманих на n/log2n сум окремих груп застосовується звичайна каскадна схема

Модифікований варіант каскадної схеми підсумовування спроектований для вирішення проблем паралельного виконання. Цей варіант може використовувати паралельні акумулятори або інші методи розпаралелювання операцій додавання, які дозволяють виконувати декілька операцій одночасно. Однак цей модифікований підхід може призвести до проблем, пов'язаних з точністю обчислень через конкуренцію за ресурси та потенційні проблеми з округленням у великих вимірах.

18. У чому відмінність показників прискорення та ефективності для розглянутих варіантів каскадної схеми підсумовування?

відмінність полягає у тому, що модифікована каскадна схема дає можливість отримати більше прискорення через оптимальне розпаралелювання операцій в групах, але при цьому вона вимагає більшої кількості процесорів для досягнення максимальної ефективності.

19. У чому полягає паралельний алгоритм обчислення всіх часткових сум послідовності числових значень?

Чим більша кількість процесорів, тим швидше виконання даного алгоритму, так як його можна розпаралелити за каскадною схемою.

20. Як формулюється закон Амдаля? Який аспект паралельних обчислень дозволяє врахувати цей закон?

Закон Амдаля показує, що навіть якщо тільки частка P програми може бути паралельною, загальне прискорення обмежене і залежить від цієї частки та кількості процесорів. З іншими словами, навіть при використанні безлічі процесорів, загальне прискорення обмежується тією частиною програми, яка не може бути паралельною. Закон Амдаля дозволяє врахувати той аспект паралельних обчислень, що вказує на те, як важливо оптимізувати самі програми для паралельного виконання. Якщо значуща частина програми може бути паралельною (велике значення P), то використання

більше процесорів може призвести до значного загального прискорення. Однак навіть невелике обмеження в паралельних можливостях програми (невелике значення P) може значно обмежити прискорення, яке можна досягти за допомогою паралельних обчислень.

21. Які припущення використовуються для обґрунтування закону Густавсона-Барсіса?

Паралельний алгоритм називають масштабованим, якщо при зростанні числа процесорів він забезпечує збільшення прискорення при збереженні постійного рівня ефективності використання процесорів.

22. Як визначається функція ізоефективності?

Якщо складність розв'язуваної задачі є фіксованою (T1=const), то при зростанні числа процесорів ефективність, як правило, буде спадати за рахунок зростання накладних витрат T0

При фіксації числа процесорів ефективність використання процесорів можна поліпшити шляхом підвищення складності розв'язуваної задачі T1

При збільшенні числа процесорів в більшості випадків можна забезпечити певний рівень ефективності за допомогою відповідного підвищення складності вирішуваних завдань.

23. Який алгоритм є масштабованим? Наведіть приклади методів з різним рівнем масштабованості.

Метод кінцевих різниць є більш масштабованим, ніж метод прямокутників для обчислення числа ПІ

3. Контрольні питання

1. Які основні характеристики використовуються для оцінки топології мережі передачі даних? Наведіть значення характеристик для конкретних типів комунікаційних структур (повний граф, лінійка, решітка та ін.).

1. **Пропускна здатність (Bandwidth)**: Кількість даних, яку можна передати через мережу за одиницю часу. Вимірюється у бітах на секунду (bps) або їх похідних.
2. **Затримка (Latency)**: Час, який потрібно для передачі сигналу від відправника до отримувача. Зазвичай вимірюється в мілісекундах (ms) або мікросекундах (µs).
3. **Масштабованість (Scalability)**: Можливість мережі збільшувати кількість підключених пристроїв чи вузлів без великих втрат продуктивності.
4. **Надійність (Reliability)**: Ймовірність, з якою мережа може надійно передавати дані без помилок чи втрат.
5. **Топологічна структура (Topology)**: Розміщення та зв'язки між вузлами мережі.

Значення цих характеристик залежать від конкретної топології. Ось порівняльні значення для декількох типів комунікаційних структур:

1. **Повний граф (Complete Graph)**:
   - **Пропускна здатність**: Висока (залежить від кількості вузлів).
   - **Затримка**: Низька (зазвичай пряма комунікація між будь-якими двома вузлами).
   - **Масштабованість**: Низька (складно масштабувати з ростом вузлів).
   - **Надійність**: Висока (можливість забезпечити багатоступеневу комунікацію).
2. **Лінійка (Linear Topology)**:
   - **Пропускна здатність**: Середня (залежить від швидкості з'єднань між вузлами).
   - **Затримка**: Середня (при збільшенні кількості вузлів може зростати затримка).
   - **Масштабованість**: Висока (легко додавати нові вузли).
   - **Надійність**: Висока (проста топологія, менше можливостей для збоїв).
3. **Решітка (Grid Topology)**:
   - **Пропускна здатність**: Залежить від розмірів решітки та швидкості з'єднань між вузлами.
   - **Затримка**: Залежить від відстані між вузлами.
   - **Масштабованість**: Середня (складніше додавати нові вузли без перепроектування всієї мережі).
   - **Надійність**: Висока (зазвичай існує багато альтернативних маршрутів для комунікації).

2. Які основні методи застосовуються при маршрутизації переданих даних по мережі?

При ***статичному*** методі маршрути розробляються і вводяться заздалегідь до програми роботи всіх вузлів мережі і не залежать від трафіку. При цьому оговорюються резервні варіанти, які включаються за розкладом або в екстрених випадках [81].

При ***динамічному*** управлінні маршрутизація здійснюється за допо-могою змінної таблиці маршрутизації. Маршрутизатор може сам визначати нові шляхи або модифікувати інформацію про старі. При цьому корекція даних про маршрути між станціями і збір інформації для прийняття рішень можуть бути проведені централізованими і децентралізованими методами [81].

При ***централізованому*** управлінні рішення про вибір маршруту приймається центром управління, наприклад сервером, та повідомляється усім вузлам, що знаходяться на даному маршруті.

При ***децентралізованому*** управлінні рішення про вибір маршруту кожним вузлом приймається автономно.

При ***змішаному*** управлінні рішення про вибір маршруту прий-мається вузлом з урахуванням рекомендацій центру управління.

3. У чому полягають основні методи передачі даних? Наведіть для цих методів аналітичні оцінки часу виконання?

1. **Провідна передача (Wired Transmission)**:
   - **Опис**: Використовує провідні кабелі для передачі даних між пристроями в мережі.
   - **Аналітична оцінка часу виконання**: Зазвичай майже миттєва, оскільки швидкість передачі даних через провід дуже велика.
2. **Безпровідна передача (Wireless Transmission)**:
   - **Опис**: Використовує радіохвилі чи інші безпровідні технології для передачі даних.
   - **Аналітична оцінка часу виконання**: Час передачі може бути вплинутий різними факторами, включаючи відстань між пристроями, перешкоди в сигналі та швидкість передачі даних. Зазвичай вимірюється в мілісекундах чи мікросекундах.
3. **Пакетна передача (Packet Switching)**:
   - **Опис**: Дані розбиваються на пакети, які незалежно пересилаються через мережу та збираються на приймачі.
   - **Аналітична оцінка часу виконання**: Залежить від розміру пакетів, швидкості мережі та затримки між відправленням та отриманням кожного пакету.
4. **Великомасштабна передача даних (Big Data Transmission)**:
   - **Опис**: Передача великих обсягів даних, часто вимагає використання спеціалізованих протоколів та алгоритмів для оптимізації передачі.
   - **Аналітична оцінка часу виконання**: Залежить від обсягу даних, пропускної здатності мережі та оптимізації алгоритмів передачі.

4. Які операції передачі даних можуть бути виділені в якості основних?

1. **Відправлення (Send)**: Передача даних з відправника до отримувача або через мережу.
2. **Отримання (Receive)**: Приймання даних від відправника або через мережу.
3. **Запит (Request)**: Відправлення запиту для отримання певних даних від іншого пристрою або сервера.
4. **Відповідь (Response)**: Відправлення відповіді на запит, що надійшов від іншого пристрою або клієнта.
5. **Зчитування (Read)**: Отримання даних з пам'яті або зберіганого ресурсу.
6. **Запис (Write)**: Запис даних в пам'ять або зберігальний ресурс.
7. **Оновлення (Update)**: Зміна або модифікація існуючих даних.
8. **Видалення (Delete)**: Видалення даних або ресурсу.
9. **Пересилання (Forward)**: Пересилання даних з одного вузла мережі на інший.
10. **Обробка (Processing)**: Обробка даних, включаючи їх аналіз, обчислення та перетворення.
11. **Публікація (Publish)**: Розміщення даних або ресурсу в мережі чи на сервері для загального доступу.

5. У чому полягають алгоритми виконання передачі даних від одного процесора всім процесорам мережі для топологій кільця, решітки та гіперкуба? Наведіть оцінки тимчасової трудомісткості для цих алгоритмів.

Топологія Кільця:

1. **Алгоритм "Обійди Кільце"**:
   - **Опис**: Кожен процесор передає дані своєму сусіду по годинниковій стрілці до відправлення даних отримувачу. Дані обходять кільце, поки не досягнуть отримувача.
   - **Оцінка Тимчасової Трудомісткості**: O(N), де N - кількість процесорів у кільці.

Топологія Решітки (Двовимірна):

1. **Алгоритм "Змійка" (Snake Algorithm)**:
   - **Опис**: Процесори передають дані один одному в рядках або стовпцях решітки. Перша половина процесорів відправляє дані праворуч, а друга половина - ліворуч (або вгору та вниз).
   - **Оцінка Тимчасової Трудомісткості**: O(N), де N - кількість процесорів у решітці.

Топологія Гіперкуба:

1. **Алгоритм "Пересилання за бітами" (Bit-Exchange Algorithm)**:
   - **Опис**: Процесори обмінюються даними з іншими процесорами, які мають одиницю в спільному біті адреси. Пошук бітів відбувається в порядку значень бітів.
   - **Оцінка Тимчасової Трудомісткості**: O(log N), де N - кількість процесорів у гіперкубі.

6. У чому полягають алгоритми виконання передачі даних від всіх процесорів всім процесорам мережі для топологій кільця, решітки та гіперкуба? Наведіть оцінки тимчасової трудомісткості для цих алгоритмів.

7. У чому полягають можливі алгоритми виконання операції редукції? Який з алгоритмів є найкращим за часом виконання?

1. **Алгоритм "Дерево" (Tree-based Reduction)**: У цьому методі процесори згруповані у дерево, і кожен вузол обчислює редукційну операцію для своїх дочірніх вузлів. Результати переадресовуються вгору по дереву до кореня, який відправляє остаточний результат.
2. **Алгоритм "Ланцюг" (Chain-based Reduction)**: Процесори вибираються у лінійний ланцюг, і кожен процесор передає свій результат наступному процесору, який потім обчислює редукційну операцію для двох значень і пересилає його наступному процесору. Ця операція повторюється до останнього процесора, який містить остаточний результат.
3. **Алгоритм "Бітова Обмін" (Bitwise Exchange-based Reduction)**: Процесори обмінюються своїми результатами за допомогою бітових операцій, таких як операція XOR. Кожен біт результата обчислюється паралельно на основі бітів від кожного процесора.
4. **Алгоритм "Дерево з рухомими даними" (Moving Data Tree-based Reduction)**: Схожий на звичайне дерево, але даними рухаються вниз та вгору через дерево, що може допомогти зменшити час затримки передачі даних.

Щодо того, який алгоритм є найкращим, це залежить від конкретних умов та характеристик системи. Алгоритми з деревами часто є ефективними, оскільки дозволяють паралельно обчислювати багато значень. Однак вибір оптимального алгоритму може також вимагати урахування додаткових факторів, таких як вартість комунікацій та латентність мережі.

8. У чому полягає алгоритм виконання операції циклічного зсуву?

Циклічний зсув вліво (Left Circular Shift):
При циклічному зсуві вліво біти числа зсуваються на певну кількість позицій уліво, і біти, які виходять за лівий край, повертаються на правий кінець числа.
Наприклад, якщо у нас є 8-бітове число **11011010** і ми виконуємо циклічний зсув вліво на 2 позиції, то результат буде **01101001**.
Циклічний зсув вправо (Right Circular Shift):
При циклічному зсуві вправо біти числа зсуваються на певну кількість позицій управо, і біти, які виходять за правий край, повертаються на лівий кінець числа.
Наприклад, якщо у нас є 8-бітове число **11011010** і ми виконуємо циклічний зсув вправо на 3 позиції, то результат буде **01011011**.

9. У чому полягає корисність використання логічних топологій? Наведіть приклади алгоритмів логічного представлення структури комунікаційної мережі?

1. **Ефективне Управління Ресурсами**: Логічні топології дозволяють розподіляти обчислювальні та комунікаційні ресурси ефективно, що може підвищити продуктивність та швидкодію мережі.
2. **Легкість Управління та Розширення**: Логічні топології можуть бути легко змінювані та розширювані без фізичних змін у мережі, що робить їх дуже гнучкими для адаптації до змінних потреб.
3. **Покращення Надійності**: Застосування логічних топологій може покращити надійність мережі, так як вони можуть вміщувати резервні шляхи та механізми відновлення.
4. **Зменшення Затрат на Кабелі та Інфраструктуру**: Оптимальне розташування вузлів у логічних топологіях може допомогти зменшити кількість потрібних кабелів та інфраструктури, що знижує вартість побудови мережі.

Наведу приклади алгоритмів для логічного представлення структури комунікаційної мережі:

1. **Алгоритм Дерева (Tree Algorithm)**: Використовується для створення ієрархічних мереж, де є головний вузол (корінь дерева) та його підвузли. Цей метод зручний для використання в локальних мережах.
2. **Алгоритм Мережі з Діаграми Процесів (Process Diagram Network Algorithm)**: Цей алгоритм використовує діаграми процесів для моделювання та аналізу взаємодії вузлів у мережі. Кожна діаграма процесу представляє взаємодію конкретних процесів у системі.
3. **Алгоритм Розподіленої Мережі (Distributed Network Algorithm)**: Використовує розподілені алгоритми для керування та обміну інформацією між вузлами у мережі. Цей метод ефективний для великих розподілених систем.

10. У чому відмінність моделей для оцінки часу виконання операцій передачі даних в кластерних обчислювальних системах? Яка модель є більш точною? Яка модель може бути використана для попереднього аналізу тимчасової трудомісткості комунікаційних операцій?

1. **Модель "Послідовний Час Передачі" (Sequential Transfer Time Model)**: Ця модель передбачає, що комунікаційні операції відбуваються послідовно, одна за одною. Вона не враховує можливість паралельності або перекриття операцій.
2. **Модель "Масштабування Пропускної Здатності" (Bandwidth Scaling Model)**: Ця модель враховує пропускну здатність мережі та передбачає, що час передачі залежить від кількості переданих даних та ширини каналу зв'язку.
3. **Модель "Затримка-Пропускна Здатність" (Latency-Bandwidth Model)**: Ця модель поєднує затримку мережі та пропускну здатність для оцінки часу передачі. Вона враховує як час, необхідний для передачі даних, так і затримки відправлення та отримання.

4. **Модель "Часу Передачі Пакета" (Packet Transfer Time Model)**: Ця модель визначає час передачі окремих пакетів даних та може бути використана для визначення часу відправлення та отримання конкретного пакета.

Яка модель є більш точною, залежить від конкретного контексту та вимог задачі. Моделі, які враховують як пропускну здатність, так і затримку, зазвичай є більш реалістичними, адже вони враховують обидві аспекти комунікаційної інфраструктури. Для попереднього аналізу тимчасової трудомісткості комунікаційних операцій часто використовується модель "Затримка-Пропускна Здатність". Ця модель може надати більш точний огляд часу передачі даних, враховуючи як затримку, так і можливість паралельності в передачі даних.

6. Контрольні питання

1. У чому полягають вихідні припущення для можливості застосування розглянутої в розділі методики розробки паралельних алгоритмів?

– Розподіл обчислень на незалежні частини – виконання їх паралельно
– Виділення інформаційних залежностей – спільний доступ до інформації
– Масштабування наявного набору підзадач – розпаралелення незалежних частин
– Розподіл підзадач між процесорами – виконуєтсья автоматично операційною системою.

2. Які основні етапи проектування та розробки методів паралельних обчислень?

Розподіл підзадач між процесорами
Розподіл обчислень на незалежні частини
Масштабування наявного набору підзадач
Виділення інформаційних залежностей

3. Як визначається модель "підзадачі-повідомлення"?

Модель "підзадачі-повідомлення" (task-message model) є популярним підходом у паралельних та розподілених обчисленнях для організації обчислювальних завдань та комунікації між різними частинами системи. Вона полягає в наступному:

Підзадачі (Tasks): Це окремі обчислювальні задачі, які мають бути виконані. Підзадачі можуть бути незалежними або залежними одна від одної в залежності від конкретної обчислювальної задачі.

Повідомлення (Messages): Це дані, які передаються між підзадачами або обчислювальними вузлами для обміну інформацією. Повідомлення можуть містити результати підзадач, інструкції для обчислення або будь-яку іншу інформацію, необхідну для виконання обчислень.

Керування (Task Management): Модель "підзадачі-повідомлення" також включає механізм для керування підзадачами, розподілу роботи між обчислювальними вузлами, збору та обробки результатів тощо.

Ця модель дозволяє розподіляти обчислювальні завдання між багатьма вузлами (частинами системи) та ефективно керувати комунікацією між ними. Вона особливо корисна у паралельних і розподілених системах, де обчислення можуть бути розділені між багатьма обчислювальними вузлами або серверами.

5. Як визначається модель "процеси-канали"?

Модель "процеси-канали":
– **Процес** - виконувана на процесорі програма, використовує для своєї роботи частину локальної пам'яті процесора і містить ряд операцій прийому / передачі даних для організації інформаційної взаємодії між виконуваними процесами паралельної програми,
– **Канал** передачі даних - черга повідомлень, в яку один або декілька процесів можуть відправляти дані що пересилаються і з якої процес-адресат може витягувати повідомлення :

• канали виникають динамічно в момент виконання першої операції прийому / передачі з каналом,
• канал може відповідати одній або декільком командам прийому / передачі даних різних процесів,
• ємність каналу необмежена,

• операції прийому повідомлень можуть призводити до блокувань (запитувані дані ще не були відправлені процесами-джерелами)

1. Які основні вимоги повинні бути забезпечені при розробці паралельних алгоритмів?
   – Досягнення рівномірного завантаження процесорів,
   – Забезпечення низького рівня інформаційної взаємодії окремих підзадач

2. У чому полягають основні дії на етапі виділення підзадач?
   Розділення завдання на підзадачі: Першою дією є аналіз вихідного завдання з метою ідентифікації частин, які можна обробити паралельно. Завдання розбивається на менші підзадачі, які можуть бути виконані окремими обчислювальними вузлами або потоками.
   Визначення залежностей між підзадачами: Важливо встановити, чи є залежності між підзадачами. Деякі підзадачі можуть вимагати результатів інших підзадач, перш ніж їх можна виконати. Це важливо для визначення порядку виконання підзадач і може впливати на схему синхронізації.
   Визначення об'єму даних для обробки: Підзадачі можуть вимагати доступу до певних даних. Важливо визначити, які дані кожна підзадача потребує для виконання своєї роботи та як ці дані будуть доступні.
   Визначення ресурсів: Потрібно визначити, які обчислювальні ресурси (наприклад, процесори, ядра, вузли) будуть виділені для кожної підзадачі. Це включає в себе призначення обчислювальних ресурсів та забезпечення доступу до них.
   Розробка стратегії паралельного виконання: Після розділення завдання на підзадачі, потрібно визначити, які підзадачі будуть виконуватися паралельно, а які можуть бути виконані послідовно або з іншими залежними підзадачами. Також потрібно визначити механізми синхронізації та комунікації між підзадачами.
   Оптимізація розподілу завдань: На цьому етапі можна оптимізувати розподіл підзадач між обчислювальними вузлами або потоками з метою забезпечення балансу завдань та мінімізації надмірної комунікації.
   Розробка алгоритмів синхронізації та комунікації: На цьому етапі розробляються механізми синхронізації та обміну даними між підзадачами. Це включає в себе вибір і реалізацію відповідних примітивів синхронізації, таких як блокування, семафори, бар'єри тощо, а також розробку механізмів передачі даних через канали чи інші засоби.

3. Які основні дії на етапі визначення інформаційних залежностей?
   Аналіз алгоритму: Ретельно проаналізуйте ваш алгоритм, визначте всі можливі інформаційні залежності між різними частинами алгоритму. Це включає в себе ідентифікацію змінних, їхніх значень та використання в різних частинах коду.
   Визначення читачів і письменників: Визначте, які частини коду (підзадачі, фрагменти алгоритму) читають дані та які записують дані. Це допоможе вам визначити, які залежності виникають між підзадачами та даними.
   Визначення типів залежностей: Встановіть типи інформаційних залежностей. Це може включати в себе залежності від даних (коли результат однієї підзадачі використовується іншою), залежності по управлінню (коли порядок виконання підзадач важливий), а також можливі залежності від ресурсів.
   Визначення рівнів залежностей: Встановіть рівні залежностей між підзадачами та даними. Деякі залежності можуть бути строго послідовними, в той час як інші можуть бути більш незалежними.
   Оцінка впливу залежностей на продуктивність: Розгляньте вплив інформаційних залежностей на продуктивність алгоритму. Деякі залежності можуть впливати на можливість паралельної обробки, тоді як інші можуть створювати обмеження.

Визначення можливостей оптимізації: Розгляньте можливості для оптимізації алгоритму з урахуванням інформаційних залежностей. Це може включати в себе перерозподіл завдань, застосування методів асинхронного обчислення та інші техніки.
Розробка стратегії керування залежностями: Після визначення інформаційних залежностей, розробіть стратегію для керування ними. Це включає вибір механізмів синхронізації, розробку алгоритмів для зменшення залежностей та оптимізації виконання.
Планування та відладка: Плануйте розподіл завдань та виконання на основі інформаційних залежностей. Після розробки алгоритму слід провести тестування та відладку, щоб переконатися, що залежності обробляються коректно.

4. У чому полягають основні дії на етапі масштабування наявного набору підзадач?

Оцінка потреб у ресурсах: Проведіть аналіз та оцінку потреб у ресурсах, таких як обчислювальні вузли, процесори, пам'ять, мережеві з'єднання тощо. Для масштабування необхідно зрозуміти, скільки додаткових ресурсів потрібно для оптимального виконання підзадач.
Оптимізація підзадач і даних: Перегляньте ваші підзадачі та дані, щоб визначити, як їх можна оптимізувати для більш ефективного виконання на більшому обсязі ресурсів. Це може включати в себе покращення алгоритмів, уникнення зайвої синхронізації та зменшення комунікації між підзадачами.
Розподіл завдань і даних: Розробіть стратегію розподілу завдань і даних між новими ресурсами. Це включає в себе вибір, які підзадачі будуть виконуватися на кожному ресурсі, та як дані будуть розподілені та синхронізовані.
Синхронізація і комунікація: Використовуйте відповідні механізми синхронізації та комунікації для забезпечення коректного та ефективного обміну даними між підзадачами на різних ресурсах.
Управління ресурсами: Забезпечте ефективне управління ресурсами під час масштабування, включаючи автоматичне збільшення та зменшення ресурсів за потребою.
Моніторинг продуктивності та масштабованості: Постійно відстежуйте продуктивність та масштабованість системи і вживайте заходів для покращення, якщо це необхідно.

9. У чому полягають основні дії на етапі розподілу підзадач по процесорах обчислювальної системи?

Розбиття завдання на підзадачі: Першою дією є розділення великого завдання на менші підзадачі, які можуть бути виконані паралельно. Це включає в себе ідентифікацію незалежних частин завдання, які можна виконувати окремими процесами або потоками.
Вибір процесорів для виконання підзадач: Розгляньте доступні процесори та виберіть, на яких процесорах (ядрах) або вузлах будуть виконуватися різні підзадачі. Вибір процесорів може бути зроблений з урахуванням їхньої доступності, продуктивності та специфікацій завдання.
Визначення ресурсів для кожної підзадачі: Для кожної підзадачі визначте, які ресурси (пам'ять, обчислювальну потужність, вхідні та вихідні дані) будуть необхідні для її виконання. Це важливо для забезпечення належного виконання підзадач.
Розробка механізмів комунікації: Якщо підзадачі взаємодіють або потребують обміну даними, розробіть механізми комунікації між ними. Це може включати в себе використання синхронізації, обмін даними через спільну пам'ять, через мережу або інші методи комунікації.
Визначення порядку виконання підзадач: Визначте порядок, в якому підзадачі будуть виконуватися на процесорах. Це може включати в себе послідовну виконання підзадач, виконання з паралельними обчисленнями або інші сценарії.

Встановлення механізмів синхронізації: Для вирішення ситуацій, де підзадачі можуть конфліктувати або взаємодіяти, розробіть механізми синхронізації. Це важливо для уникнення конфліктів та недопущення гонок.

10. Як відбувається динамічне управління розподілом обчислювального навантаження за допомогою схеми "менеджер - виконавець"?

– Для управління розподілом навантаження в системі виділяється окремий процесор-менеджер, якому доступна інформація про всі наявні Підзадача,
– Решта процесори системи є виконавцями, які? Для отримання обчислювального навантаження звертаються до процесора-менеджеру.
– Породжувані в ході обчислень нові підзадачі передаються назад процесору-менеджеру і можуть бути отримані для вирішення при наступних зверненнях процесорів-виконавців,
– Завершення обчислень відбувається в момент, коли процесори-виконавці завершили вирішення всіх переданих їм підзадач, а процесор-менеджер не має яких-небудь обчислювальних робіт для виконання.

1. Який метод паралельних обчислень був розроблений для вирішення гравітаційної завдання N тіл?

Очевидний алгоритм вирішення задачі N тіл полягає у розгляді на кожному кроці моделювання всіх пар об'єктів фізичної системи та виконанні для кожної одержуваної пари всіх необхідних розрахунків

12. Який спосіб виконання операції узагальненого збору даних є більш ефективним?

– Метод 1: Обмін даними здійснюється в ході послідовності кроків, на кожному з яких всі наявні підзадачі розбиваються попарно і обмін даними здійснюється між парами підзадач, що утворилися (N-1 ітерація).
– Метод 2: Перший крок методу виконується так само, як у методі 1 - після виконання цього кроку підзадачі будуть утримувати свої дані і дані підзадач, з якими вони утворювали пари. Як результат, на другому кроці пари підзадач можуть бути утворені для обміну даними відразу про два тіла фізичної системи. Після завершення другого кроку кожна підзадача міститиме відомості про чотири тіла системи і т.д. Тим самим, загальна кількість кроків для виконання всіх необхідних обмінів є рівним log2N