Тема 6: Моделі паралельних обчислень

Питання:

1. Паралелізм даних

2. Паралелізм задач

3. Етапи розробки паралельного алгоритму

Найрозповсюдженішим підходом до розпаралелення обчислень і обробки даних є підхід, що базується на моделях паралелізму даних і паралелізму задач. В основі кожного підходу лежить розповсюдження обчислювальної роботи між доступними користувачу процесорами паралельного комп'ютера. Причому виникають *проблеми* забезпечення рівномірного завантаження процесорів і ефективної організації обміну інформації між ними.

1. Паралелізм даних

Паралелізм даних базується на застосуванні однієї операції до кількох елементів масиву даних і може використовуватися при векторній і паралельній обробці.

Основними ознаками підходу є:

- обробкою даних керує одна програма;
- простір є глобальним (для програміста є одна єдина пам'ять, а деталі структури даних доступні до пам'яті і міжпроцесорного обміну для програміста скриті);
- слабка синхронізація обчислень на паралельних процесорах (кожен процесор виконує один той же фрагмент програми, але нема гарантії, що в даний момент часу на всіх процесорах виконується одна і таж машинна програма);
- паралельні операції над елементами масиву виконується на всіх допустимих даній програмі процесорах.

Базовими операціями для даної моделі паралелізму є: операції керування даними, операції над масивами та їх фрагментами, умовні операції, операції приведення, зсуву, сканування, пересилання даних.

Для програмування в моделі паралелізму даних використовуються спеціалізовані мови програмування, здебільшого різновидності мови FORTRAN. А сам програміст – слабо впливає на процес і результати роботи

2. Паралелізм задач

Метод передбачає розбиття задачі на кілька відносно самостійних підзадач. Найефективніше дана модель реалізовується на *MIMD* структурі, де кожен процесор виконує свою програму (FORTRAN чи C). Оскільки в даному випадку програмісту самому необхідно контролювати як розподіл даних між процесорами і різними під задачами так і організацію обміну даними між ними, то підхід є більш трудоємкісним від попереднього.

При паралелізмі задач виникають певні *проблеми*:

- підвищена трудоємність розробки і відлагодження програм;
- на програміста покладається вся відповідальність за збалансоване завантаження програм;
- програмісту необхідно мінімізувати обмін даними між процесорами.
- підвищена небезпека виникнення тупикових ситуацій.

Проте забезпечується висока гнучкість і свобода програмісту у виборі напрямку і особливостей організації паралельної роботи. Крім того, для даної моделі є розроблений великий клас спеціалізованих бібліотек: MPI (Message Passing Interface), PVM (Parallel Virtual Machines).

3. Етапи розробки паралельного алгоритму

Найважливішим та найважчим етапом при створенні програми є розробка алгоритму, який повинен з одного боку забезпечити ефективність використання паралельної обчислювальної системи, а з іншого – мати добру масштабованість. В загальному випадку процес створення паралельного алгоритму можна розбити на чотири кроки (див. рис.4.1).

1. *Декомпозиція*. На етапі вихідна задача аналізується, оцінюється можливість її розпаралелення. При незначному виграші від розпаралелення і великій трудоємкості розробки

паралельної програми перший крок розбиття алгоритму виявляється і останнім. Якщо ж ситуація відмінна від описаної, то задача та пов'язані з нею дані розділяються на дрібніші частини – підзадачі і фрагменти структур даних. Особливості архітектури конкретної обчислювальної системи на цьому етапі можуть не враховуватися.

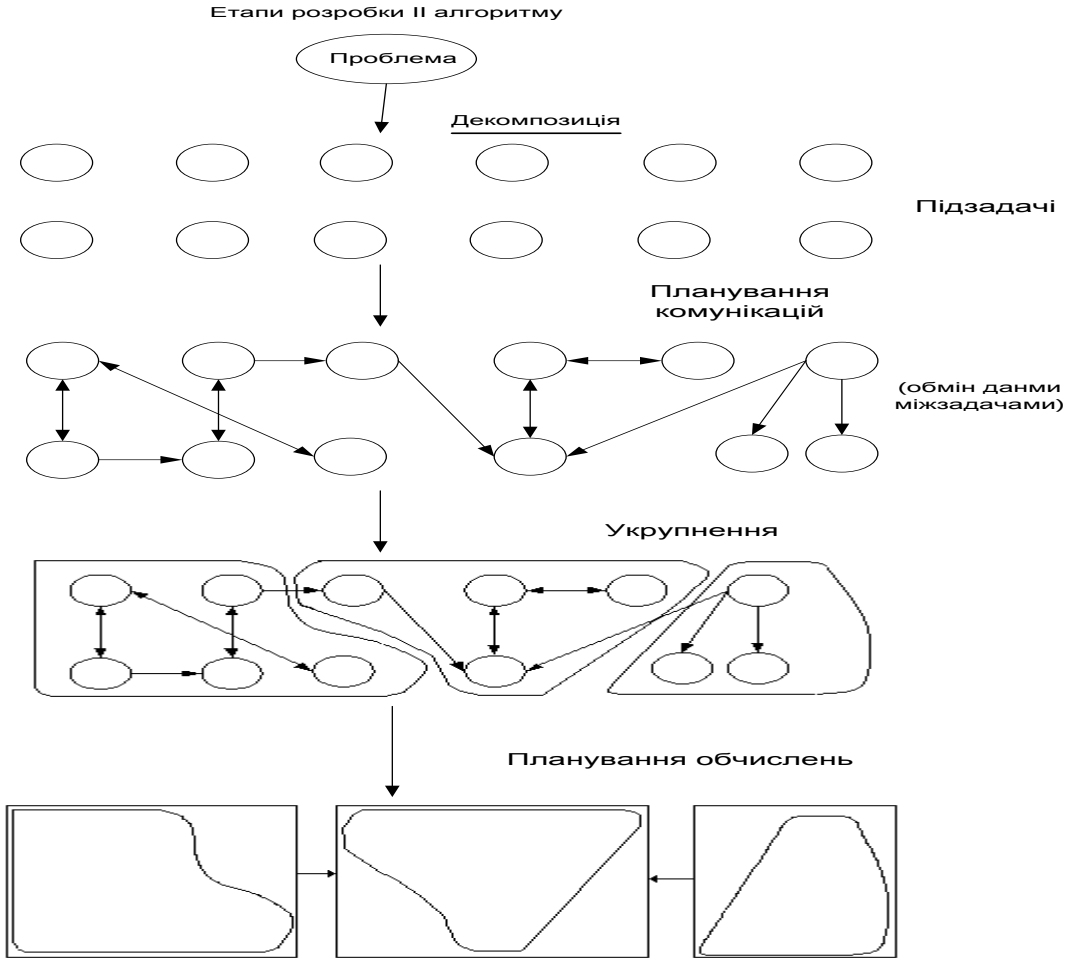


Рис.4.1. Етапи розбиття паралельного алгоритму

2. *Проектування комунікацій* (обміну даними) між підзадачами. На етапі визначаються зв'язки, необхідні для пересилання вхідних даних, а також комунікації, що необхідні для керування роботою підзадач. Обираються методи та алгоритми комунікацій.

3.*Укрупнення*. Підзадачі можуть об'єднуватися у більші блоки, якщо це дозволяє підвищити ефективність алгоритму і знизити трудоємкість розробки. Основними критеріями на даному кроці є ефективність та трудоємкість реалізації алгоритму.

4. *Планування обчислень*. На цьому кроці виконується розподіл підзадач між процесорами. Основний критерій вибору способу розміщення підзадач – ефективне використання процесорів з мінімальними затратами часу на обмін даними.

Розглянемо ці етапи детальніше.

Декомпозиція (сегментування).

На етапі визначається ступінь можливого розпаралелення задачі. Іноді декомпозиція поставленої задачі природним чином випливає з самої задачі тому є очевидною, іноді – ні. Чим менший розмір підзадач, отриманих в результаті декомпозиції, чим більша їх кількість, тим гнучкішим виявляється паралельний алгоритм і тим легше забезпечити рівномірне завантаження процесорів обчислювальної системи. Надалі, можливо доведеться укрупнити алгоритм, оскільки слід прийняти до уваги інтенсивність обміну даними та інші фактори.

Сегментувати можна як обчислювальний *алгоритм, так і дані*. Застосовуються різні варіанти декомпозиції.

В методі *декомпозиції даних спочатку* сегментуються *дані*, а *потім алгоритм* їх обробки. Дані розбиваються на сегменти приблизно однакового розміру. З фрагментами даних пов'язуються операції їх обробки, з яких і формуються підзадачі. Визначаються необхідні правила обміну даними. Перекриття частин, на які розбивається задача, повинне бути мінімальним. Це дозволяє уникнути дублювання обчислень. Схема розбиття може в подальшому уточнюватися. Іноді, якщо це необхідно для зменшення кількості обмінів, допускається збільшення ступені перекриття фрагментів задачі.

При декомпозиції даних спочатку аналізуються структури даних найбільшого розміру, або ті, до яких найчастіше відбувається звертання. На різних стадіях розрахунків можуть використовуватися різні структури даних, тому можуть бути необхідними динамічні, тобто такі, що міняються в часі, схеми декомпозиції цих структур.

В методі *функціональної декомпозиції спочатку* сегментується обчислювальний *алгоритм, а потім* під цю схему підбирається схема декомпозиції *даних*. Успіх у цьому випадку не завжди гарантовано, оскільки схема може вимагати багатьох додаткових пересилань даних. Використання методу корисне при відсутності структур даних, які б могли бути розпаралелені очевидним чином. Функціональна декомпозиція відіграє важливу роль і як метод структурування програм. Вона може виявитися корисною при моделюванні складних систем, що складаються з множини простих підсистем, зв'язаних між собою набором інтерфейсів.

Розмір блоків, з яких складається паралельна програма, може бути різним. В залежності від розміру блоків, алгоритм може мати різну "*зернистість*". Її виміром в найпростішому випадку є кількість операцій у блоці. Виділяють три ступені зернистості: *дрібнозернистий, середньоблоковий* та *великоблоковий*. Зернистість пов'язана з рівнем паралелізму.

Паралелізм на рівні команд найбільш дрібнозернистий. Його масштаб менше ніж 20 команд на блок. Кількість підзадач, що паралельно виконуються – від однієї до кількох тисяч., при чому середній масштаб паралелізму складає приблизно з п'яти команд на блок.

Наступний рівень - паралелізм на рівні циклів. Переважно, цикл містить не більше ніж 500 команд. Якщо ітерації циклу незалежні, вони можуть виконуватися, наприклад, за допомогою конвеєра або на векторному процесорі. Це також дрібнозернистий паралелізм.

Паралелізм на рівні підзадач – середньоблоковий. Розмір блоку – до 2000 команд. При реалізації такого паралелізму слід враховувати можливі міжпроцедурні залежності. Вимоги до комунікацій менші, ніж у випадку паралелізму на рівні команд.

Паралелізм на рівні програм (задач) – великоблоковий. Він означає виконання незалежних програм на паралельному комп'ютері і повинен підтримуватися операційною системою.

Обмін даними через спільні/розподілені змінні використовується на рівнях дрібнозернистого і середньоблокового паралелізму, а на великоблоковому – засобами передачі повідомлень.

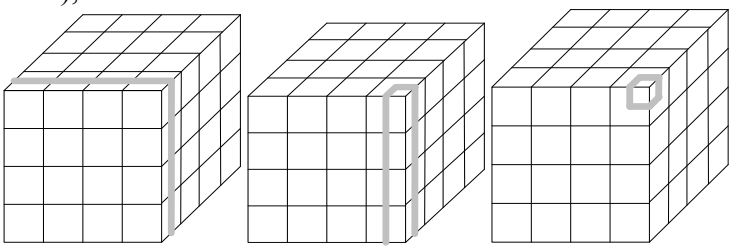
Досягнути ефективності роботи паралельної програми можна, якщо збалансувати зернистість алгоритму і затрати часу на обмін даними.

Частини програми можуть виконуватися паралельно, лише якщо вони незалежні. Є такі *види незалежності*:

- *незалежність за даними* (дані, які обробляються однією частиною програми не модифікуються іншою її частиною);
- *незалежність за керуванням* (послідовність виконання частин програми може бути визначена лише під час виконання програми - наявність залежності по виконанню наперед визначає порядок виконання);
- *незалежність за ресурсами* (забезпечується достатньою кількістю комп'ютерних ресурсів - об'єм пам'яті, кількість функціональних вузлів та ін.);
- *незалежність за виводом* виникає, якщо дві підзадачі виконують запис в одну і ту ж змінну. А *залежність за вводом/виводом*, якщо оператори вводу/виводу двох чи декількох підзадач звертаються до одного файлу(чи змінної).

Тобто, *дві підзадачі можуть виконуватися паралельно, якщо вони незалежні за даними, за керуванням і за операціями вводу/виводу*.

Приклад декомпозиції даних для сіткових методів (застосовуються в інженерних і наукових розрахунках) наведений на рис.4.2. Декомпозиція в цьому випадку може бути одновимірною (великоблочна, рис.4.2.а), двовимірною (рис.4.2.б), тривимірною (дрібнозерниста, рис.4.2.в),



а) б) в

Рис.4.2. *Приклад декомпозиції даних*

На практиці частіше використовується декомпозиція даних.

Для забезпечення ефективності декомпозиції необхідно дотримуватись таких *рекомендацій*:

- кількість підзадач після декомпозиції повинна приблизно на порядок переважати кількість процесорів (забезпечується гнучкість на наступних етапах);
- необхідно уникати лишніх обчислень і пересилань даних (в іншому випадку програма буде погано масштабованою і не дозволить досягнути високої ефективності при розв'язанні задач великого обсягу);
- підзадачі повинні бути приблизно однакового розміру (легше забезпечити збалансованість завантаження процесора);
- в ідеалі: сегментація повинна бути такою, щоб з збільшенням об'єму задачі кількість підзадач також зростала, при збереженні постійного розміру однієї підзадачі (забезпечує хорошу масштабованість).

Добитися ефективної роботи другої програми можна шляхом збалансування "зернистості" алгоритму і затрати часу на обмін даними.

<u>Проектування комунікацій</u>

Оскільки підзадачі не можуть бути абсолютно незалежними, наступним етапом розробки паралельного алгоритму *є проектування обмінів даними* між ними, яке полягає в визначенні структури каналів зв'язку і повідомлень, якими повинні обмінюватися підзадачі.

Канали зв'язку можуть програмуватися явно і неявно. Якщо на першому етапі виконувалась декомпозиція даних, тоді проектувати комунікації дуже важко. При функціональній декомпозиції проектування комунікацій простіше – вони відповідають потокам даних між підзадачами.

Комунікації класифікуються за такими *ознаками: розміщення, структурованості, за відношенням до зміни в часі, за відношенням до засобів синхронізації.*

За *розміщенням* комунікації поділяються на:

- *локальні* (кожна підзадача зв'язана з невеликою кількістю інших підзадач);
- *глобальні* (кожна підзадача зв'язана з великою кількістю інших підзадач).

За ознакою *структурованості* комунікації поділяються на:

- *структуровані* (кожна підзадача і підзадачі, що зв'язані з нею утворюють регулярну структуру);
- *неструктуровані* (підзадачі зв'язані довільним графом).

За відношенням до зміни в часі комунікації поділяються на:

- *статичні* (схема комунікації з часом не змінюється);
- *динамічні* (схема комунікації змінюється в процесі виконання програми).

За відношенням до засобів синхронізації комунікації є:

- *синхронні* (відправник і отримувач даних координують обмін між собою).
- *асинхронні* (обмін даними не координується).

Рекомендації з проектування комунікацій:

- *кількість* комунікацій у підзадачах повинна бути приблизно *однаковою* (в іншому випадку буде погана масштабованість);
- ефективніше використовувати *локальні* комунікації;
- комунікації повинні бути *паралельними.*

Крім того, при проектуванні комунікацій необхідно уникати *тупикових ситуацій*, які можуть бути зв'язані з неправильною послідовністю обміну даними між процесами (Наприклад, перший процес очікує дані від другого, другий від третього, а останньому потрібні результати роботи першого).

Обмін повідомленнями може бути реалізований по різному: за допомогою потоків, міжпроцесорних комунікацій, тощо. Найрозповсюдженіший спосіб програмування комунікацій – використання бібліотек, що реалізують обмін повідомленнями, наприклад, бібліотеки *PVM* і *MPI* дозволяють створити паралельні програми для різних платформ. *CORBA* (Common Object Reguest Broker Architecture) визначає протокол взаємодії між процесами, незалежний від мови програмування і операційної системи. Для опису інтерфейсу використовується *декларативна* мова *IDL* (Interface Definition Language).

Укрупнення

На етапі, на відміну від попередніх, враховується архітектура обчислювальної системи. При цьому часто приходиться об'єднувати задачі, які були отримані на попередніх етапах, для того, щоб їх кількість дорівнювала кількості процесорів.

Метою укрупнення є збільшення зернистості обчислень і комунікацій, збереження гнучкості і зменшення вартості розробки.

Загальні *вимоги:*

- зниження затрат на комунікації;
- якщо при укрупнені приходиться дублювати обчислення чи дані, це не повинно приводити до втрат продуктивності і масштабованості програми;
- результуючі задачі повинні мати приблизно однакову трудоємність;
- збереження масштабованості;
- збереження можливості паралельного виконання;
- зменшення вартості і трудоємності розробки.

Планування обчислень

На цьому етапі визначається де і на якому процесорі виконується підзадача. Основним критерієм ефективності є мінімізація часу виконання програми. Проблема планування обчислень легко вирішується при використанні систем з розділеною пам'яттю.

В алгоритмах основаних на функціональній декомпозиції, часто створюється множина дрібних "короткоживучих" підзадач. В цьому випадку застосовується *алгоритми планування виконання задач (tasks scheduling)*, які розподіляють підзадачі по незагружених роботою процесорах.

Планування виконання задач полягає в організації їх доступу до ресурсів. Порядок представлення доступу визначається алгоритмом. Наприклад:

- планування по принципу кругового обслуговування (всім процесорам задається однаковий час на виконання підзадачі);
- метод збалансованого завантаження – оснований на обміну біжучого завантаження процесорів;
- планування виконання задач.

Структури, що виконують такі задачі діляться на ієрархічні і децентралізовані; в децентралізованих – головна задача відсутня.

Важливою проблемою є вибір методу планування (статичний чи динамічний) і збалансованість завантаження. При розв'язанні складних задач динамічні методи планування

простіші, але продуктивність програм менша. Динамічно збалансоване завантаження може бути ефективно реалізоване, якщо враховано такі підходи:

- якщо кожен процесор виконує одну підзадачу, тривалість виконання всієї програми буде визначатися часом виконання найдовшої підзадачі;
- оптимальна продуктивність досягається, якщо всі задачі мають приблизно однаковий розмір;
- забезпечувати збалансованість роботи процесорів може програміст, але є засоби автоматичної підтримки збалансованої роботи;
- збалансованість може бути забезпечена шляхом завантаження одного процесора кількома задачами.

Є різні алгоритми *збалансованого завантаження в методах декомпозиції даних,* наприклад:

- *рекурсивна дихотомія.* Використовується для розбиття області на підобласті, яким відповідає приблизно однакова трудоємність обчислень; комунікації звернені до мінімуму;
- *рекурсивна координатна дихотомія.* Застосовується до нерегулярних сіток. Ділення виконується на кожному кроці вздовж того виміру, по якому сітка має найбільшу довжину;
- *рекурсивна дихотомія графа.* Використовується для нерегулярних сіток (мінімізується кількість ребер);
- *ймовірнісні методи.* Задачі розміщаються на випадково вибраних процесорах.
- *циклічні планування.* Є різновидністю ймовірнісних методів. Вибирається певна схема нумерації підзадач, і в кожний N – процесор завантажується N – задача.

Вправи і завдання до теми №6

1. Дайте порівняльну характеристику моделям паралелізму даних і паралелізму задач.
2. Розробіть структуру паралельного алгоритму виконання функції $y = \sum_{i=1}^{20} (x_i + z_i)$.
3. Як переваги забезпечує використання збалансованого завантаження процесорів?
4. На якому етапі розробки паралельного алгоритму необхідно враховувати параметри процесорів?
5. Чи завжди на етапі укрупнення головною вимогою є “Зниження затрат на комунікації”?