Лекція 2. Основи паралельних обчислювальних процесів.

План лекції:

1. Вимоги до паралельних программ
2. Парадигми паралельного програмування.
   2.1. Паралелізм даних
   2.2. Паралелізм завдань
3. Моделі паралельного програмування
   3.1. Завдання/канал.
   3.2. Модель спільної пам'яті.
   3.3. Передача повідомлень
4. Продуктивність паралельних обчислень.
   4.1. Закон Амдала.
   4.2. Особливості закону Амдала.
   4.3. Закон Густавсона.

1. Вимоги до паралельних програм.

Паралельні обчислення — спосіб організації комп'ютерних обчислень, при якому програми розробляються як набір взаємодіючих обчислювальних процесів, що працюють паралельно (одночасно).

Основна складність при проектуванні паралельних програм — забезпечити правильну послідовність взаємодій між різними обчислювальними процесами, а також координацію ресурсів, що розділяються між процесами.

Паралельна програма –це безліч взаємодіючих паралельних процесів. Основною метою паралельних обчислень є прискорення вирішення обчислювальних завдань. Паралельні програми володіють наступними особливостями:

1) здійснюється управління роботою безлічі процесів;

2) організовується обмін даними між процесами;

3) втрачається детермінізм поведінки через асинхронність доступу до даних;

4) переважають нелокальні і динамічні помилки;

5) з'являється можливість тупикових ситуацій;

6) виникають проблеми масштабованості програми і балансування завантаження обчислювальних вузлів.

Основними рисами моделі паралельного програмування є більш висока продуктивність програм, застосування спеціальних прийомів програмування та, як наслідок, більш висока трудомісткість програмування, проблеми з перенесенням програм. Паралельна модель не володіє властивістю унікальності. В паралельної моделі програмування з'являються проблеми, незвичні для програміста, який звик займатися послідовним програмуванням. Серед них: керування роботою безлічі процесорів,

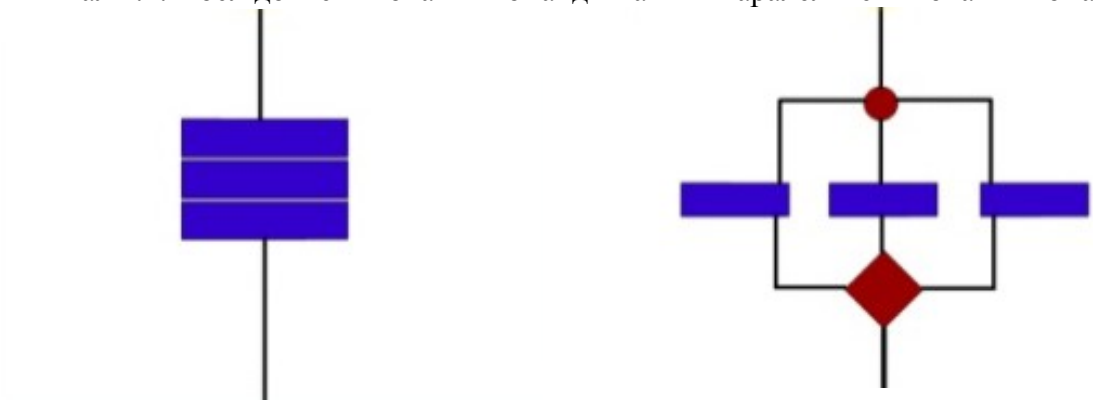
організація межпроцессорных пересилок даних і т. д. Можна сформулювати чотири фундаментальних вимоги до паралельних програм:

- паралелізм,
- універсальність,
- локальність,
- модульність.

Ідея розпаралелювання обчислень заснована на тому, що більшість завдань можуть бути розділені на набір менших завдань, які можуть бути вирішені одночасно.

Паралельна програма використовує множинність обчислювальних одиниць (процесорів, ядер, GPU і т. д.) для прискорення обчислень. Ми поділяємо роботу між одночасно працюючими обчислювачами в надії, що виграш від паралелізму перевершить додаткові витрати на нього. Прискорення обчислення — єдина мета паралелізму.

Мал 2.1. Послідовне виконання команд та Паралельне виконання команд



Ідея локальності полягає в тому, що коли процес виконується, він рухається від однієї локальності до іншої. Локальність - набір сторінок, які активно використовуються разом. Програма складається з декількох різних локальностей, які можуть перекриватися. Наприклад, коли викликана процедура, вона визначає нову локальність, що складається з інструкцій процедури, її локальних змінних, і безлічі глобальних змінних. Після її завершення процес залишає цю локальність, але може повернутися до неї знову. Таким чином, локальність визначається кодом і даними програми. Зауважимо, що модель локальності - принцип, покладений в основу роботи кеша. Якби доступ до будь-яких типів даних був випадковим, кеш був би даремним.

Початкова орієнтація та налаштування на широкий клас задач предметної галузі, в якій працює прикладної фахівець, є однією з ключових особливостей ППП (пакетів прикладних програм). Поєднання в одному пакеті безлічі різноманітних моделей і алгоритмів досягається шляхом використання принципу модульної організації функціонального наповнення пакету. Модуль, який входить до складу ППП, видається, як

правило, у вигляді автономної програмної одиниці, написаної традиційному мовою програмування, що забезпечує рішення деякої підзадачі. Передбачається, що взаємодія модулів здійснюється тільки на рівні їх вхідних і вихідних параметрів. Використання принципу модульності дозволяє замінити написання програми (в традиційному розумінні) її конструюванням з готових програмних блоків великого розміру. Розширення класу задач, розв'язуваних пакетом, може досягатися за рахунок підключення до ППП новостворюваних модулів.

Відношення прискорення або ефективності обсягу використовуваних ресурсів або розміру задачі характеризує те, наскільки раціонально програма здатна використовувати паралельну обчислювальну систему. Для цього вводиться поняття масштабованості паралельної програми. Масштабованість – властивість програми, що визначає залежність її прискорення або ефективності, одержуваних на обчислювальній системі, від обсягу використаних для цього ресурсів і розміру задачі. Масштабованість є однією з основних характеристик паралельної програми. Вона дозволяє отримати уявлення про роботу паралельної програми на даної обчислювальної системи.

Зазначимо, що ідеальна паралельна програма володіє наступними властивостями:

- паралельно виконувані гілки повинні мати приблизно однакову довжину;
- повністю виключені простої із-за очікування даних, передачі управління і виникнення конфліктів при використанні загальних ресурсів;
- обмін даними повністю сумісний з обчисленнями.

Збільшення ступеня ефективності паралелізму (зменшення часових витрат на накладні витрати) досягається наступними способами:

- укрупненням одиниць розпаралелювання;
- зменшенням складності алгоритмів генерації паралельних процедур (підпрограм);
- початковою підготовкою пакета різних варіантів вихідних даних;
- розпаралелюванням алгоритмів генерації паралельних процедур (підпрограм).

2. Парадигми паралельного програмування.

Найбільш поширеними підходами до паралельного виконання обчислень і обробки даних є підходи, засновані на моделях паралелізму даних і паралелізму задач. В основі підходу лежить розподіл обчислювальної роботи між доступними користувачеві процесорами паралельного комп'ютера. При цьому доводиться вирішувати різноманітні проблеми. Насамперед, це рівномірне завантаження процесорів, оскільки якщо основна обчислювальна робота буде лягати тільки на частину з них, зменшиться і виграш від

розпаралелювання. Інша не менш важлива проблема — ефективна організація обміну інформацією між завданнями. Якщо обчислення виконуються на високопродуктивних процесорах, завантаження яких досить рівномірне, але швидкість обміну даними при цьому низька, більша частина часу буде витрачатися даремно на очікування інформації, необхідної для подальшої роботи завдання. Це може істотно знизити швидкість роботи програми.

2.1. Паралелізм даних

Основна ідея підходу, заснованого на паралелізм даних, — застосування однієї операції відразу декільком елементам масиву даних. Різні фрагменти такого масиву обробляються на векторному редакторі чи на різних процесорах паралельної машини. Векторизація або розпаралелювання в цьому випадку найчастіше виконуються вже під час трансляції - перекладу вихідного тексту програми в машинні команди.

Основні особливості розглянутого підходу:

- паралельною обробкою даних управляє одна програма;
- простір імен є глобальним, тобто для програміста існує одна єдина пам'ять, а деталі структури даних, доступу до пам'яті і межпроцессорного обміну даними від нього приховані;
- слабка синхронізація паралельних обчислень на процесорах, тобто виконання команд на різних процесорах відбувається, як правило, незалежно і тільки іноді проводиться узгодження виконання циклів або інших програмних конструкцій — їх синхронізація. Кожен процесор виконує один і той же фрагмент програми, але немає гарантії, що в заданий момент часу на всіх процесорах виконується одна і та ж машинна команда;
- паралельні операції над елементами масиву виконуються одночасно на всіх доступних даній програмі процесорах.

Таким чином, в рамках даного підходу від програміста не потрібно великих зусиль по векторизації або розпаралелювання обчислень. Навіть при програмуванні складних обчислювальних алгоритмів можна використовувати бібліотеки підпрограм, спеціально розроблених з урахуванням конкретної архітектури комп'ютера і оптимізованих для цієї архітектури.

Підхід, заснований на паралелізм даних, базується на використанні в програмах базового набору операцій.

Реалізація моделі паралелізму даних вимагає підтримки паралелізму на рівні транслятора. Таку підтримку можуть забезпечувати:

препроцесори, які використовують існуючі послідовні транслятори та спеціалізовані бібліотеки, з реалізаціями паралельних алгоритмічних конструкцій;

предтранслятои, які виконують попередній аналіз логічної структури програми, перевірку залежностей і обмежену паралельну оптимізацію;

розпаралелюючі транслятори, які виявляють паралелізм у вихідному коді програми і виконують його перетворення у паралельні конструкції. Для того щоб спростити перетворення, у вихідний текст програми можуть додаватися спеціальні директиви трансляції.

2.2. Паралелізм завдань

Метод програмування, заснований на паралелізмі завдань, передбачає розбиття обчислювальної задачі на кілька відносно самостійних підзадач. Кожне завдання виконується на своєму процесорі. Комп'ютер при цьому являє собою MIMD-машину. Для кожної підзадачі пишеться своя власна програма на звичайній мові програмування, найчастіше це FORTRAN або C. Підзадачі повинні обмінюватися результатами своєї роботи, отримувати вихідні дані. Практично такий обмін здійснюється викликом процедур спеціалізованої бібліотеки. Програміст при цьому може контролювати розподіл даних між різними процесорами і різними підзадачами, а також обмін даними. У цьому випадку, очевидно, потрібна додаткова робота для того, щоб забезпечити ефективне виконання різних завдань. Порівняно з підходом, заснованим на паралелізмі даних, цей підхід більш трудомісткий, з ним пов'язані такі проблеми:

- підвищена трудомісткість як розробки програми, так і її налагодження;
- на програміста лягає вся відповідальність за рівномірне і збалансоване завантаження процесорів паралельного комп'ютера;
- програмісту доводиться мінімізувати обмін даними між завданнями, т. к. витрати часу на пересилку даних зазвичай відносно великі;
- підвищена небезпека виникнення тупикових ситуацій, коли надіслана однією програмою посилка з даними не приходить до місця призначення.

3. Моделі паралельного програмування.

Сукупність прийомів програмування, структур даних, що відповідають архітектурі гіпотетичного комп'ютера, призначеного для виконання певного класу алгоритмів, називається моделлю програмування. Нагадаємо, що алгоритм — це кінцевий набір правил, розташованих у певному логічному порядку і дозволяє виконавцю вирішувати будь-яку конкретну задачу з деякого класу однотипних задач.

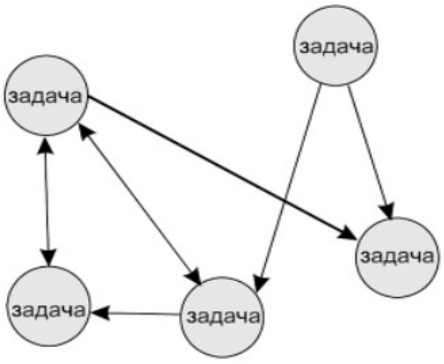
Опис алгоритму є ієрархічним, його складність залежить від ступеня його деталізації. На самому верхньому рівні ієрархії знаходиться завдання в цілому. Її можна вважати деякою узагальненої операцією. Нижній рівень ієрархії включає примітивні операції, машинні команди.

Традиційною вважається послідовна модель програмування. У цьому випадку у будь-який момент часу виконується тільки одна операція і тільки над одним елементом даних. Послідовна модель універсальна. Її основними рисами є застосування стандартних мов програмування (для вирішення обчислювальних задач це, зазвичай, FORTRAN 90/95 і C/C++), хороша переносимість програм і невисока продуктивність.

3.1. Модель задача/канал.

У найпростішій моделі паралельного програмування, вона називається моделлю завдання/канал (мал. 2.2), програма складається з декількох завдань, пов'язаних між собою каналами комунікації і виконуються одночасно. Кожне завдання складається з послідовного коду та локальної пам'яті. Можна вважати, що завдання — це віртуальна фон-нейманівська машина. Набір вхідних і вихідних портів визначає її інтерфейс з оточенням.

Канал – це черга повідомлень-посилок з даними. Завдання може помістити в чергу своє повідомлення або, навпаки, видалити повідомлення, прийнявши дані, які містяться в ньому. Кількість завдань може змінюватися в процесі виконання. Крім зчитування і запису даних у локальну пам'ять кожна задача може посилати і приймати повідомлення, породжувати нові завдання і завершувати їх виконання. Операція відправки даних асинхронна, вона завершується відразу. Операція прийому синхронна — вона блокує виконання завдання до тих пір, поки не буде отримано повідомлення. Кількість каналів також може змінюватися в процесі виконання програми.



Мал. 2.2. Модель задача/канал

Незалежність результату виконання паралельної програми від зображення на конкретну архітектуру забезпечується тим, що завдання взаємодіють між собою за допомогою універсального механізму. Він не залежить від того, як розподіляються завдання, тому і результат виконання програми не повинен залежати від способу розподілу.

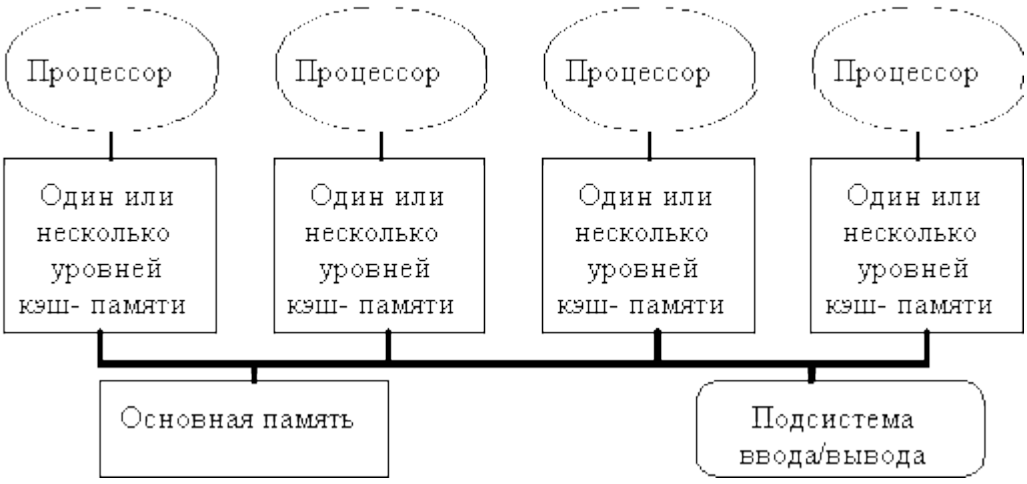
У моделі завдання/канал зберігаються і переваги модульного програмування. Частини модульної програми створюються окремо і об'єднуються на завершальному етапі розробки програми. Взаємодія між модулями забезпечується добре визначеними інтерфейсами, а самі модулі можуть модифікуватися окремо, незалежно від інших модулів. Завдання є природним будівельним блоком в модульній конструкції програми.

3.2. Модель спільної пам'яті.

У моделі спільної пам'яті завдання звертаються до загальної пам'яті, маючи спільний адресний простір і виконуючи операції зчитування/запису. Управління доступом до пам'яті здійснюється за допомогою різних механізмів, таких, наприклад, як семафори. В рамках цієї моделі не потрібно описувати обмін даними між завданнями в явному вигляді. Це спрощує програмування. Разом з тим особливу увагу доводиться приділяти дотриманню детермінізму.

Найпростіший спосіб створити багатопроцесорний обчислювальний комплекс зі спільною пам'яттю — взяти кілька процесорів, з'єднати їх загальною шиною між собою і з оперативною пам'яттю. Це простий, але не найкращий спосіб, оскільки, якщо один процесор приймає команду або передає дані, всі інші процесори змушені переходити в режим очікування. Це призводить до того, що, починаючи з деякого числа процесорів, швидкодія такої системи перестає збільшуватися при додаванні нового процесора.

Ми вже з'ясували, що частково поліпшити картину може застосування кеш-пам'яті для зберігання команд. При наявності локальної, тобто такої, що належить даному процесору, кеш-пам'яті, наступна необхідна йому команда з великою ймовірністю буде перебувати в кеш-пам'яті. В результаті цього зменшується кількість звернень до шини і швидкодія системи зростає.



Мал.. 2.3. Модель спільної пам'яті

Кеш-пам'ять діє наступним чином. Якщо центральний процесор звертається до оперативної пам'яті, спочатку запит на необхідні дані надходить в кеш-пам'ять. Якщо вони там вже знаходяться, виконується швидке пересилання даних в регістр процесора, в іншому випадку дані зчитуються з оперативної пам'яті і розміщуються в кеш-пам'яті, а з неї вже завантажуються в регістр.

При використанні в багатопроцесорних обчислювальних системах виникає проблема когерентності кеш-пам'яті. Вона полягає в тому, що якщо двом або більше процесорам знадобилося значення однієї і тієї ж змінної, воно буде зберігатися у вигляді декількох копій в кеш-пам'яті всіх процесорів. Один із процесорів може змінити це значення в результаті виконання своєї команди і воно буде передано в оперативну пам'ять комп'ютера. Але в кеш-пам'яті інших процесорів все ще зберігається старе значення. Отже, необхідно забезпечити своєчасне оновлення даних в кеш-пам'яті всіх процесорів комп'ютера.

Проблема кеш-когерентності може вирішуватися програмним шляхом — на рівні транслятора і операційної системи. Транслятор, наприклад, може визначати моменти безпечної синхронізації кеш-пам'яті при виконанні програми. Інший підхід заснований на апаратному вирішенні проблеми кеш-когерентності. У цьому випадку застосовуються спеціальні протоколи, кеш-пам'ять використовується більш ефективно, все відбувається "прозоро" для програміста. Протоколи каталогів реалізують збір і облік інформації про копії даних у кеш-пам'яті. Ця інформація зберігається в спеціальній області оперативної пам'яті — каталозі. Запити перевіряються за допомогою каталогу, потім виконуються необхідні пересилання даних. Використання даного підходу ефективно у великих системах зі складними схемами комунікації. Протоколи стеження розподіляють "відповідальність" за когерентність кешей між контролерами кеш-пам'яті. Контролер

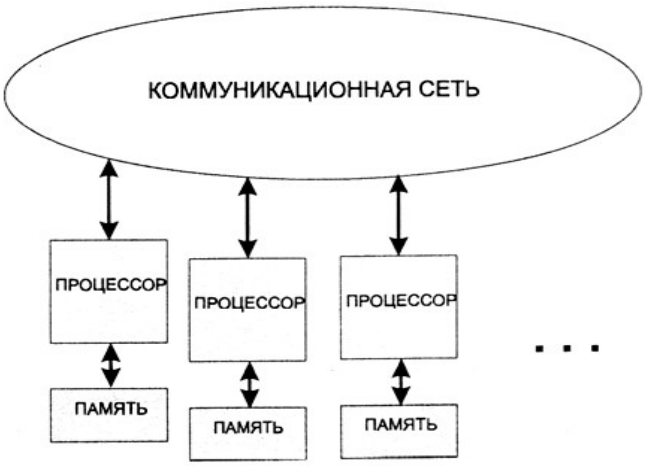
визначає зміну вмісту кеш-пам'яті і передає інформацію про це іншим модулям кеш-пам'яті. Очевидно, в цьому випадку зростає обсяг переданих даних. У деяких системах використовується адаптивна суміш обох підходів.

Є різні реалізації поділюваної пам'яті. Це, наприклад, поділювана пам'ять з дискретними модулями пам'яті. Фізична пам'ять складається з декількох модулів, хоча віртуальний адресний простір залишається загальним. Замість загальної шини в цьому випадку використовується перемикач, який направляє запити від процесора до пам'яті. Такий перемикач може одночасно обробляти кілька запитів до пам'яті, тому, якщо всі процесори звертаються до різних модулів пам'яті, швидкодія зростає.

3.3. Передача повідомлень

Передача повідомлень — одна з найпоширеніших моделей паралельного програмування. Програми, написані в рамках цієї моделі, як і програми в моделі завдання/канал, при виконанні породжують кілька завдань. Кожному завданню присвоюється свій унікальний ідентифікатор, а взаємодія здійснюється за допомогою відправки і прийому повідомлень. Можна вважати, що дана модель програмування відрізняється від моделі завдання/канал тільки механізмом передачі даних. Повідомлення надсилається не в канал, а певній задачі.

У моделі передачі повідомлень нові завдання можуть створюватися під час виконання паралельної програми, кілька завдань можуть виконуватися на одному процесорі. Однак на практиці при запуску програми найчастіше створюється фіксоване число однакових завдань, і це число залишається незмінним під час виконання програми. Така різновид моделі називається SPMD-моделлю (Single Program Multiple Data — одна програма, масив даних), оскільки кожна задача містить один і той же код, але обробляє різні дані.



Мал. 2.4. Модель передачі повідомлень

Системи передачі повідомлень використовують канали і блокування повідомлень, що створює додатковий трафік на шині і вимагає додаткових витрат пам'яті для організації черг повідомлень. У сучасних процесорах можуть бути передбачені спеціальні комутатори (кроссбары) з метою зменшення впливу обміну повідомленнями на час виконання завдання. Існує великий вибір математичних теорій для вивчення та аналізу систем з передачею повідомлень. Найбільш відома - модель акторів.

Передача повідомлень може бути ефективно реалізована на симетричних мультипроцесорах як з поділюваною кеш-пам'яттю, так і без неї. У паралелізму зі спільною пам'яттю і з передачею повідомлень різна продуктивність, так як час, необхідний на перемикання підзадач в системі з передачею повідомлень, менше ніж час, необхідний на доступ до комірки пам'яті, проте передача самих повідомлень вимагає додаткових ресурсів, на відміну від виклику процедур.

При розпаралелюванні важливо враховувати не тільки формальний паралелізм структури алгоритму, але і те, що операції обміну даними в паралельних ЕОМ відбуваються, як правило, значно повільніше арифметичних.

4. Продуктивність паралельних обчислень.

Найважливішим критерієм ефективності паралельного програмування є швидкодія програми, її продуктивність. На продуктивність впливають різні фактори. Це технологія виконання апаратної частини (в тому числі електронних компонентів), архітектура обчислювальної системи, методи управління ресурсами, ефективність паралельного алгоритму, особливості структури даних, ефективність мови програмування, кваліфікація програміста, ефективність транслятора і т. д. Час виконання програми залежить від часу доступу до головної і зовнішньої пам'яті, кількості операцій введення та виведення, завантаженості операційної системи.

Процесор управляється тактовим генератором, що виробляє керуючі імпульси фіксованої тривалості — такти. Зворотна тривалості імпульсу величина називається частотою. Виконання кожної машинної команди вимагає кількох тактів. Кількість тактів на команду (CPI — Cycles Instruction Per) характеризує трудомісткість і тривалість команди. В різних класах програм різне середнє значення CPI, тому воно може служити чисельною характеристикою програми.

Процесорний час, необхідний для виконання програми, можна визначити за формулою:

$T=Ni*CPI*t,$

де Ni — кількість машинних команд у програмі, а t — тривалість такту.

Швидкодія процесора вимірюється в MIPS (Million Instructions Per Second). MIPS зворотно пропорційна CPI.

4.1. Закон Амдала

У загальному випадку структура інформаційного графа алгоритму займає проміжне положення між крайніми випадками повністю послідовного і повністю паралельного алгоритму. У цій структурі є фрагменти, які допускають одночасне виконання на декількох функціональних пристроях -- це паралельна частина програми. Є і фрагменти, які повинні виконуватися послідовно і на одному пристрої — це послідовна частина програми.

З допомогою інформаційного графа можна оцінити максимальне прискорення, якого можна досягти при розпаралелюванні алгоритму там, де це можливо. Припустимо, що програма виконується на машині, архітектура якої ідеально відповідає структурі інформаційного графа програми. Нехай час виконання алгоритму на послідовній машині T1, причому Ts — час виконання послідовної частини алгоритму, а Tp — паралельної. Очевидно:

$T1 = Ts + Tp.$

При виконанні тієї ж програми на ідеальній паралельній машині, N незалежних гілок паралельної частини розподіляються по одній на N процесорів, тому час виконання цієї частини зменшується до величини Tp / N, а повний час виконання програми складе:

$T2=Ts+ Tp/N.$

Коефіцієнт прискорення, який показує, у скільки разів швидше програма виконується на паралельній машині, ніж на послідовній, визначається формулою:

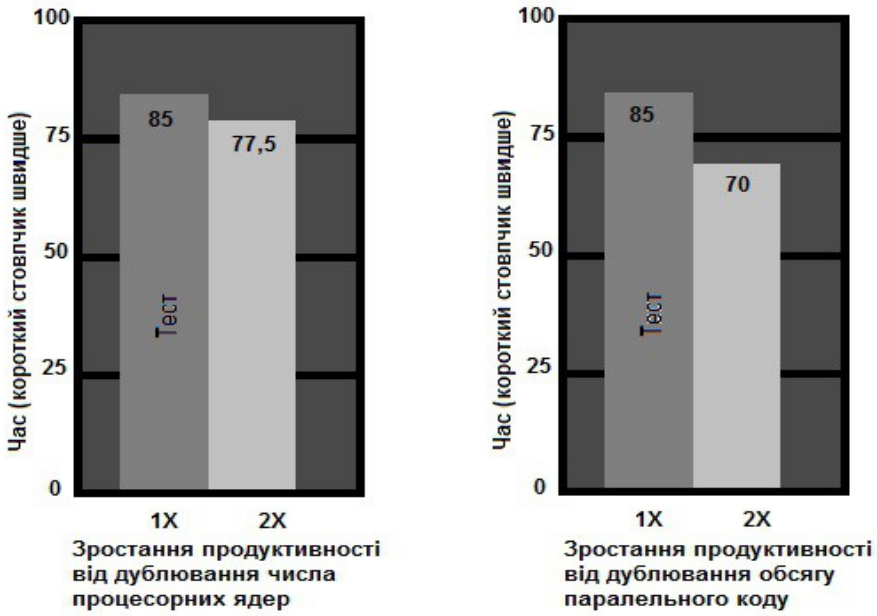
$X= T1/T2 = (Ts + Tp) / (Ts + Tp/N) = 1/(S + P/N),$

де S= Ts / (Ts + Tp) і P= Tp / (Ts + Tp) — відносні частки послідовної і паралельної частин (S+ P= 1). Залежність коефіцієнта прискорення від числа процесорів і ступеня паралелізму алгоритму (відносної частки паралельної частини) носить назву закону Амдала.

Для програм (алгоритмів) з невеликим ступенем паралелізму використання великого числа процесорів не дає скільки-небудь значного виграшу в швидкодії. Якщо ж ступінь паралелізму досить велика, коефіцієнт прискорення може бути великим. Починаючи з деякого значення, збільшення числа процесорів дає тільки невеликий приріст продуктивності. На практиці, коли доводиться приймати до уваги кінцевий час обміну даними, при збільшенні кількості процесорів може спостерігатися навіть падіння продуктивності, оскільки збільшується кількість обмінів.

Ступенем паралелізму називають кількість процесорів, які використовуються в кожен момент часу для виконання програми. Ступінь паралелізму може змінюватися в процесі виконання програми. Це, взагалі кажучи, змінна величина, яка залежить, зокрема, від наявності та доступності ресурсів, тому іноді вводять різні середні характеристики паралелізму.

Дослідження показали, що потенційний паралелізм в наукових і технічних прикладних програмах може бути дуже великим, до сотень і тисяч команд на такт, однак реальний ("досяжний") паралелізм значно менше — 10-20.



Мал. 2.5. Теоретичне порівняння збільшення кількості процесорних ядер та паралелізму реалізації (додатку)

Закон Амдала (англ. Amdahl'slaw, іноді також Закону Амдала-Уера) — ілюструє обмеження зростання продуктивності обчислювальної системи із збільшенням кількості обчислювачів. Джин Амдал сформулював закон в 1967 році, виявивши просте по суті, але непереборне за змістом обмеження на зростання продуктивності при розпаралелюванні обчислень: «У випадку, коли завдання поділяється на кілька частин, сумарний час його виконання на паралельній системі не може бути менше часу виконання самого довгого фрагмента».[1] Згідно з цим законом, прискорення виконання програми за рахунок розпаралелювання її інструкцій на множині обчислювачів обмежена часом, необхідним для виконання її послідовних інструкцій.

4.2 Особливості закону Амдала.

Закон Амдала описує максимальний теоретичний виграш в продуктивності паралельного рішення по відношенню до кращого послідовного вирішення. Закон Амдала описується наступною математичною формулою:

$$S_n = \frac{1}{\alpha + \frac{1-\alpha}{n}}$$

Де S_n у скільки разів можна прискорити обчислення (прискорення), n - кількість процесорів (ядер), α - частка послідовно обчислюваного коду ($\alpha \neq 0$).

Закон Амдала, незважаючи на те, що він не враховує багатьох факторів, накладає обмеження на максимально досяжну ефективність паралельного алгоритму.

Припустимо, наприклад, що $\alpha = \frac{1}{3}$, тобто дві третини операцій в алгоритмі можуть виконуватися паралельно, а третина - ні. Тоді прискорення $S_n < 3$. Таким чином, незалежно від кількості процесорів (ядер) і навіть при ігноруванні всіх витрат на підготовку даних не можна прискорити рішення завдання більш, ніж у три рази.

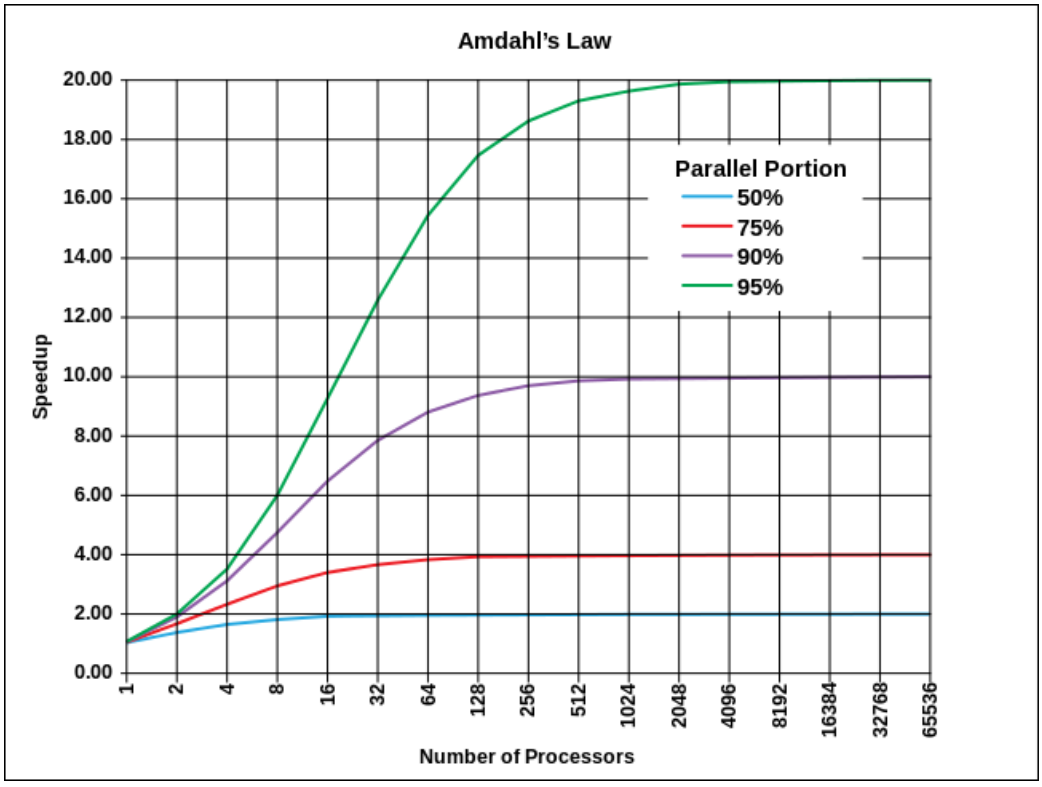
4.3. Закон Густавсона.

Закон Густавсона - Барсиса (1988р) оцінює максимально допустиме прискорення виконання паралельної програми, в залежності від кількості одночасно виконуваних потоків обчислень і частки послідовних розрахунків. Формула Густавсона - Барсиса виглядає наступним чином:

$$S_n = n + (1 - n)\alpha$$

Деα - частка послідовних розрахунків в програмі, n- кількість процесорів.

Густавсон зауважив, що, працюючи на багатопроцесорних системах, користувачі схильні до зміни тактики вирішення задачі. Тепер зниження загального часу виконання програми поступається обсягу розв'язуваної задачі. Така зміна мети обумовлює перехід від закону Амдала до закону Густавсона. Приміром, на 100 процесорах програма виконується 20 хвилин. При переході на систему з 1000 процесорами можна досягти часу виконання близько двох хвилин. Однак для одержання більшої точності рішення має сенс збільшити обсяг розв'язуваної задачі, тобто при збереженні загального часу виконання користувачі прагнуть отримати більш точний результат. Збільшення обсягу розв'язуваної задачі призводить до збільшення частки паралельної частини, так як послідовна частина (введення/виведення, менеджмент потоків, точки синхронізації тощо) не змінюється.



Мал. 2.6. Ілюстрація закону Амдала. Прискорення програми з допомогою паралельних обчислень на декількох процесорах