Паралельні та розподілені обчислення

Модулі 1,2

«Паралельне програмування»

конспект лекцій

Освітній ступінь – «Бакалавр»

КИЇВ – 2021

Модуль 1. Багатопоточність
Лекція 1. Основи паралельних обчислень

Визначення та переваги паралелізму. Засоби для проведення паралельних обчислень. Архітектура ОС. Класифікація обчислювальних систем. Моделі послідовного і паралельного програмування. Шляхи досягнення паралелізму.

Паралелізм на рівні команд, потоків, застосунків. Основні етапи розробки паралельних програм: декомпозиція, виявлення інформаційних залежностей між підзадачами, масштабування підзадач і балансування навантаження для кожного процесора. Загальна схема розпаралелювання задач.

1. Визначення та переваги паралелізму

Загалом, під паралельними обчисленнями розуміються процеси обробки даних, в яких одночасно можуть виконуватися декілька операцій комп'ютерної системи.

Паралелізм – сукупність математичних, алгоритмічних, програмних і апаратних засобів, що забезпечують можливість *паралельного виконання задачі.*

Переваги паралелізму

- Паралельність підвищує продуктивність системи за допомогою більш ефективного витрачання системних ресурсів. Наприклад, під час очікування появи даних по мережі, обчислювальна система може використовуватися для вирішення локальних завдань.
- Паралельність підвищує відгук програми. Якщо один потік зайнятий розрахунком або виконанням якихось запитів, то інший потік може реагувати на дії користувача.
- Паралельність полегшує реалізацію багатьох застосувань. Багато програм типу "клієнт-сервер", "виробник-споживач" мають внутрішній паралелізм. Послідовна реалізація таких застосувань більше трудомістка, ніж опис функціональності кожного учасника окремо.

2. Засоби для проведення паралельних і розподілених обчислень

Для того, щоб реалізувати паралельні та розподілені обчислення, необхідні апаратні і програмні рішення.

Апаратні

засоби для проведення обчислень (обчислювальна техніка):

- обчислювальна техніка, зібрана з стандартних комплектуючих;
- обчислювальна техніка, зібрана з спеціальних комплектуючих (суперкомп’ютери);

засоби для зберігання і обробки даних.

Програмні:

- програмні засоби загального призначення (операційні системи, стандартні бібліотеки, мови програмування, компілятори, відлагоджувачі і т.п.)
- спеціальні програмні засоби: бібліотеки паралельного програмування (PVM, MPI); засоби об’єднання ресурсів (Dynamite, Globus і ін.)

Досягнення паралелізму можливе лише при виконанні наступних вимог до архітектурних принципів побудови обчислювального середовища (апаратних рішень):

- **незалежність функціонування окремих пристроїв комп'ютера** – ця вимога стосується всіх основних компонентів обчислювальної системи: пристроїв введення-виведення, процесорів, пам'яті;
- **надмірність елементів обчислювальної системи** – організація надмірності може здійснюватися в наступних основних формах:

- використання спеціалізованих пристроїв, таких, наприклад, як окремі процесори, пристрої багаторівневої пам'яті (регістри, кеш);

- дублювання пристроїв комп'ютера шляхом використання, наприклад, декількох однотипних процесорів (або ядер) або декількох пристроїв оперативної пам'яті.

3. Класифікація обчислювальних систем

3.1. Класифікація Флінна

Класифікація обчислювальних систем

Однією з найбільш поширених класифікацій обчислювальних систем є класифікація Флінна. Чотири класи обчислювальних систем виділяються у відповідність з двома вимірами - характеристиками систем: *потік команд*, які ця архітектура здатна виконати в одиницю часу (одиничний або множинний) і *потік даних*, які можуть бути оброблені в одиницю часу (одиничний або множинний).

- SISD = Single Instruction Single Data
- MISD = Multiple Instruction Single Data
- SIMD = Single Instruction Multiple Data
- MIMD = Multiple Instruction Multiple Data
- **SISD** (Single Instruction, Single Data) – системи, в яких існує один потік команд і один потік даних. До такого типу можна віднести звичайні послідовні ЕОМ;
- **SIMD (Single Instruction, Multiple Data) –** системи з одиничним потоком команд і множинним потоком даних. Подібний клас складають багатопроцесорні обчислювальні системи, в яких в кожен момент часу може виконуватися одна і та сама команда для обробки декількох інформаційних елементів; таку архітектуру мають, наприклад, багатопроцесорні системи з єдиним пристроєм управління. Цей підхід широко використовувався в попередні роки в суперкомп'ютерах (системи ILLIAC IV або CM-1 компанії Thinking Machines), останнім часом його використання обмежене, в основному, створенням спеціалізованих систем;
- **MISD** (Multiple Instruction, Single Data) – системи, в яких існує множинний потік команд і одиночний потік даних. Відносно цього типу систем немає єдиної думки: ряд фахівців вважає, що прикладів конкретних ЕОМ, відповідних даному типу обчислювальних систем, не

існує і введення подібного класу робиться для повноти класифікації; інші ж відносять до цього типу, наприклад, системи з конвеєрною обробкою даних;

- **MIMD** (Multiple Instruction, Multiple Data) – системи з множинним потоком команд і множинним потоком даних. До подібного класу відносяться більшість паралельних багатопроцесорних обчислювальних систем.

Класифікація Флінна відносить майже усі паралельні обчислювальні системи до одного класу - MIMD. Для виділення різних типів паралельних обчислювальних систем застосовується класифікація Джонсона, в якій подальший розподіл багатопроцесорних систем ґрунтується на використовуваних способах організації оперативної пам'яті в цих системах. Цей підхід дозволяє розрізняти два важливі типи багатопроцесорних систем: **multiprocessors (мультипроцесорні або системи із загальною (спільною) пам'яттю, що розділяється) і multicomputers (мультикомп'ютери або системи з розподіленою пам'яттю).**

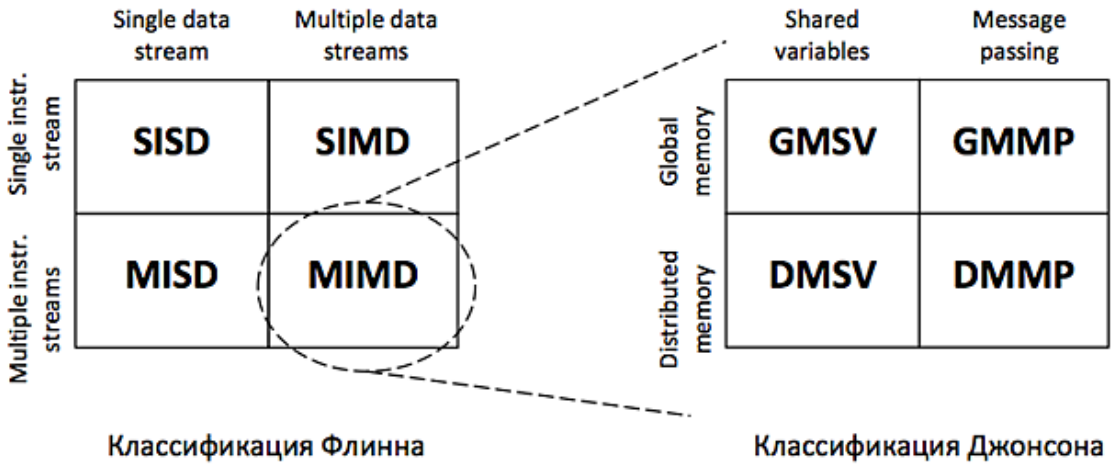


Рис. 1.1. Класифікація багатопроцесорних обчислювальних систем

Класифікація Джонсона заснована на **структурі пам'яті** (global - глобальна або distributed - розподілена) і механізмі комунікацій і синхронізації (shared variables - змінні, що розділяються, або message passing - передача повідомлень). Системи GMSV (global - memory - shared - variables) часто називаються також **мультипроцесорами з пам'яттю, що розділяється** (shared - memory multiprocessors). Системи DMMP (distributed - memory - message - passing) також звані **мультикомпьютерами з розподіленою пам'яттю** (distributed - memory multicomputers).

Далі розглянемо це більш детально.

3.2. Мультипроцесори і мультикомп'ютери

Для подальшої систематики мультипроцесорів враховується спосіб побудови *спільної пам'яті*. Перший можливий варіант – використання єдиної (централізованої) **загальної** пам'яті (shared memory) (рис. 1.2 а). Такий підхід

забезпечує однорідний доступ до пам'яті (uniform memory access або UMA) і є основою для побудови векторних паралельних процесорів (parallel vector processor або PVP) і симетричних мультипроцесорів (symmetric multiprocessor або SMP). До цього типу відносяться і багатоядерні архітектури.

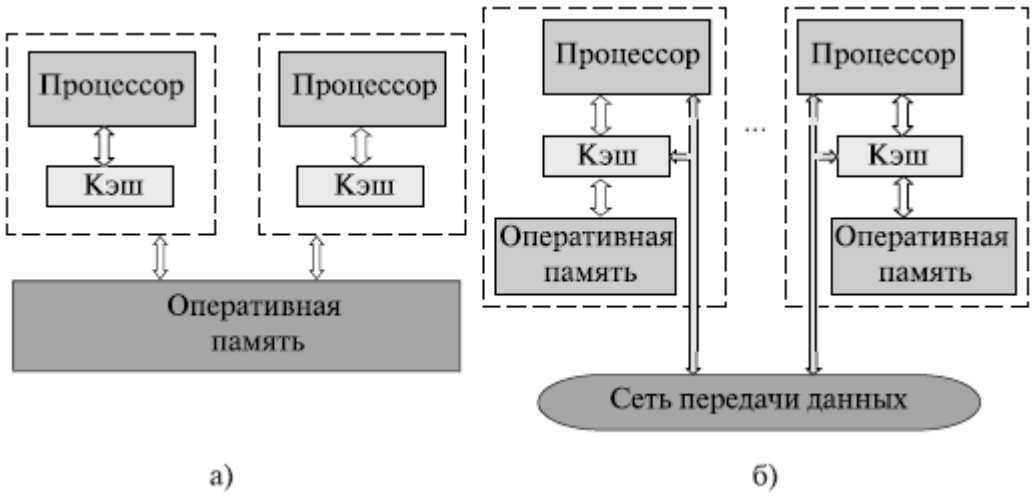


Рис. 1.2. Архітектура багатопроцесорних систем із спільною (що розділяється) пам'яттю: системи з однорідним (а) і неоднорідним (б) доступом до пам'яті

Однією з основних проблем, які виникають при організації паралельних обчислень на такого типа системах, є доступ з різних процесорів до спільних даних і забезпечення, у зв'язку з цим, однозначності вмісту різних кешів (cache coherence problem). Річ у тому, що за наявності спільних даних копії значень одних і тих самих змінних можуть виявитися в кешах різних процесорів. Якщо в такій ситуації (за наявності копій спільних даних) один з процесорів виконає зміну значення змінної, що розділяється, то значення копій в кешах інших процесорів виявляться не правильними і їх використання призведе до некоректності обчислень. Забезпечення однозначності кешів зазвичай реалізується на апаратному рівні – для цього після зміни значення спільної змінної всі копії цієї змінної в кешах оголошуються як недійсні і подальший доступ до змінної потребує обов'язкового звернення до основної пам'яті. Слід зазначити, що необхідність забезпечення однозначності призводить до деякого зниження швидкості обчислень і утруднює створення систем з великою кількістю процесорів.

Наявність спільних даних при паралельних обчисленнях призводить до необхідності *синхронізації взаємодії* одночасно виконуваних потоків команд. Так, наприклад, якщо зміна спільних даних вимагає для свого виконання деякої послідовності дій, то необхідно забезпечити таке взаємовиключення (mutual exclusion), щоб ці зміни у будь-який момент часу міг виконувати лише один командний потік. Завдання взаємовиключення і синхронізації належать до класичних проблем, і їх розгляд при розробці паралельних програм є одним з основних питань паралельного програмування.

3.2.1. Мультипроцесор з розподіленою пам'яттю

У *мультипроцесорі і в багатоядерній системі* процесори і ядра процесорів мають доступ до оперативної пам'яті, що розділяється. Процесори *спільно використовують оперативну пам'ять*.

Наявність спільних даних при виконанні паралельних обчислень призводить до необхідності синхронізації взаємодії одночасно виконуваних потоків команд. Так, наприклад, якщо зміна даних вимагає для свого виконання деякої послідовності дій, то необхідно забезпечити, щоб ці зміни у будь-який момент часу міг виконувати тільки один командний потік. Завдання взаємовиключення і синхронізації належать до класичних проблем, і їх розгляд при розробці паралельних програм є одним з основних питань паралельного програмування.

3.2.2. Мультикомп'ютери

Мультикомп'ютери (багатопроцесорні системи з розподіленою пам'яттю) вже не забезпечують спільного доступу до всієї наявної в системах пам'яті (no-remote memory access або NORMA) (рис. 1.3). При всій схожості подібної архітектури з системами з розподіленою спільною пам'яттю, мультикомп'ютери мають принципову відмінність: кожен процесор системи може використовувати лише свою локальну пам'ять, тоді як для доступу до даних, що розташовуються на інших процесорах, необхідно явно виконати операції передачі повідомлень (message passing operations). Даний підхід застосовується при побудові двох важливих типів багатопроцесорних обчислювальних систем - масивно-паралельних систем (massively parallel processor або MPP) і кластерів (clusters).

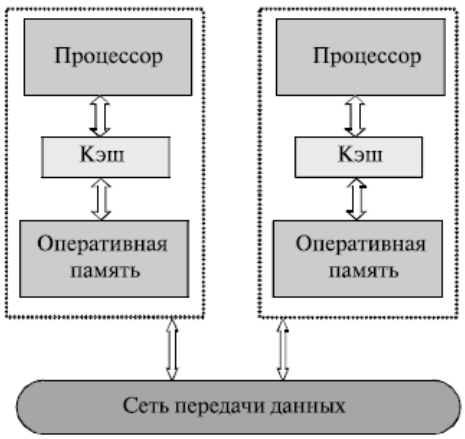


Рис. 1.3. Архітектура багатопроцесорних систем з розподіленою пам'яттю

Слід зазначити надзвичайно швидкий розвиток багатопроцесорних обчислювальних систем кластерного типу. Кластери відрізняються від звичайних мереж комп'ютерів.

Визначення (Вікіпедія)

Кластер - група комп'ютерів, об'єднаних високошвидкісними каналами зв'язку і які представляються з точки зору користувача як єдиний апаратний

ресурс. Кластер - слабо пов'язана сукупність декількох обчислювальних систем, що працюють спільно для виконання загальних застосувань, і що представляються користувачеві єдиною системою. Один з перших архітекторів кластерної технології Грегорі Пфистер дав кластеру наступне визначення: "Кластер - це різновид паралельної або розподіленої системи, яка, :

1. складається з декількох пов'язаних між собою комп'ютерів;
2. використовується як єдиний, уніфікований комп'ютерний ресурс".

Зазвичай розрізняють наступні основні види кластерів :

1. відмовостійкі кластери (High - availability clusters, HA, кластери високої доступності)
2. кластери з балансуванням навантаження (Load balancing clusters)
3. обчислювальні кластери (High perfomance computing clusters, HPC)
4. системи розподілених обчислень.

Для побудови локальної комп'ютерної мережі, як правило, використовують простіші мережі передачі даних (порядка 100 Мбіт/сек). Комп'ютери мережі зазвичай більш розосереджені, і користувачі можуть застосовувати їх для виконання яких-небудь додаткових робіт.

4. Види паралелізму на рівні програмного забезпечення

При розгляді проблеми організації паралельних обчислень слід розрізняти наступні можливі режими виконання незалежних частин програми:

- *багатозадачний режим* (режим розділення часу), який припускає, що кількість підзадач (*процесів або потоків* одного процесу) більше, ніж кількість процесорів. Цей режим є ***псевдопаралельним*,** коли *активним* (виконуваним) може бути один процес, а усі інші процеси (потоки) знаходяться в стані очікування своєї черги на використання процесора. Використання режиму розділення часу може підвищити ефективність організації обчислень (наприклад, якщо один з процесів не може виконуватися через очікування даних, що вводяться, процесор може бути задіяний для виконання іншого, готового до виконання процесу), крім того в цьому режимі проявляються багато ефектів паралельних обчислень (необхідність взаємовиключень і синхронізації процесів та ін.).

Багатопоточність програм в операційних системах з розділенням часу застосовується навіть в *однопроцесорних системах*. Наприклад, в Windows –застосунках багатопоточність підвищує відгук застосунку - якщо основний потік зайнятий виконанням якихось розрахунків або запитів, інший потік дозволяє реагувати на дії користувача. Багатопоточність спрощує розробку застосунку. Кожен потік може плануватися і виконуватися незалежно. Наприклад, коли користувач натискає кнопку мишки персонального комп'ютера, посилається сигнал процесу, що управляє вікном, в якому в даний момент знаходиться курсор миші. Цей процес (потік) може виконуватися і

відповідати на клацання миші. Застосунки в інших вікнах можуть продовжувати при цьому своє виконання у фоновому режимі.

- *синхронні паралельні обчислення (або паралельні обчислення)*, коли одночасно може виконуватися декілька команд обробки даних.

Цей режим обчислень називається ***справжнім паралелізмом***, але може бути забезпечений не лише за наявності декількох процесорів, але і за допомогою конвеєрних і векторних оброблювальних пристроїв.

Мета паралельних обчислень - швидко вирішити задачу або за той же час вирішити велику задачу. Приклади синхронних паралельних обчислень – задачі великого розміру:

- наукові обчислення, які моделюють і імітують такі явища, як глобальний клімат, еволюція сонячної системи або результат дії нових ліків;
- графіка і обробка зображень, включаючи створення спецефектів в кіно;
- великі комбінаторні або оптимізаційні задачі, наприклад, планування авіа перельотів або економічне моделювання.

Програми рішення таких задач вимагають ефективного використання доступних обчислювальних ресурсів системи. Число підзадач має бути оптимізоване з урахуванням числа процесорів або ядер процесорів.

- *розподілені обчислення* цей термін застосовують для вказівки паралельної обробки даних, при якій використовується декілька процесорів, досить віддалених один від одного, в яких передача даних по лініях зв'язку призводить до істотних затримок часу. Як результат, ефективна обробка даних при такому способі організації обчислень можлива лише для паралельних алгоритмів з низькою інтенсивністю потоків міжпроцесорних передач даних. Перераховані умови є характерними, наприклад, при організації обчислень в багатомашинних обчислювальних комплексах, що утворюються об'єднанням декількох окремих ПК за допомогою каналів зв'язку локальних або глобальних інформаційних мереж (клієнт-сервер, веб).

Рівні паралелізму в багатоядерній архітектурі:

Паралелізм на рівні команд (InstructionLevelParallelism, ILP) дозволяє процесору виконувати декілька команд за один такт. Залежності між командами обмежують кількість доступних для виконання команд, знижуючи об'єм паралельних обчислень. Технологія ILP дозволяє процесору переупорядкувати команди оптимальним чином з метою виключити зупинки обчислювального конвеєра і збільшити кількість команд, що виконуються процесором за один такт. Сучасні процесори підтримують певний набір команд, які можуть виконуватися паралельно.

Паралелізм на рівні потоків процесу. Потоки дозволяють виділити незалежні групи команд у рамках одного процесу. Потоки підтримуються на рівні операційної системи. Операційна система розподіляє потоки процесів по ядрах процесора з урахуванням пріоритетів. За допомогою потоків програма може максимально задіювати вільні обчислювальні ресурси.

Паралелізм на рівні застосунків. Одночасне виконання декількох програм здійснюється в усіх операційних системах, що підтримують режим розділення

часу. Навіть на однопроцесорній системі незалежні програми виконуються одночасно (по черзі). Паралельність досягається за рахунок виділення кожному застосунку кванта процесорного часу.

Отже, базовими поняттями ОС при виконанні задач є ***процес і потік***.

5. Процеси і потоки

При розробці програми з одним або декількома процесорами можна реалізувати розділення алгоритму програми таким чином, щоб програма забезпечувала взаємодію з користувачем, навіть якщо вона зайнята іншими діями. Одним із способів, що забезпечує взаємодію програми і користувача в процесі обробки програмою інших подій, є використання *декількох потоків* виконання.

Під *процесом* розуміють абстракцію ОС, яка об'єднує все необхідне для виконання однієї програми в певний момент часу.

Можна сказати, що *процес* – це програма під час її виконання.

Для успішного виконання програми потрібні певні ресурси. До них належать:

-ресурси, необхідні для послідовного виконання програмного коду (передусім *процесорний час*);

-ресурси, що дають можливість зберігати інформацію, яка забезпечує виконання програмного коду *(регістри процесора, оперативна пам'ять тощо*).

Ці групи ресурсів визначають дві складові частини процесу:

- послідовність виконуваних команд процесора;

- набір адрес пам'яті (адресний простір), у якому розташовані ці команди і дані для них.

Виділення цих частин виправдане ще й тим, що в рамках одного адресного простору може бути кілька паралельно виконуваних послідовностей команд, що спільно використовують одні й ті самі дані. Необхідність розмежування послідовності команд і адресного простору підводить до поняття *потоку*.

Потоком (потік керування, нитка, thread) називають набір послідовно виконуваних команд процесора, які використовують спільний адресний простір процесу.

Оскільки в системі може одночасно бути багато потоків, завданням ОС є організація перемикання процесора між ними і планування їхнього виконання.

У багатопроцесорних системах код окремих потоків може виконуватися на окремих процесорах.

Потік містить такі елементи:

- *стан процесора* (набір поточних даних із його регістрів), зокрема лічильник поточної інструкції процесора;

- *стек потоку* (ділянка пам'яті, де перебувають локальні змінні потоку й адреси повернення функцій, що викликані у його коді).

Використання декількох потоків у програмі означає внесення в нього *паралелізму (concurrency)*.

Багатопоточність (multithreading) - це спеціалізована форма багатозадачності (multitasking). В основному, виділяють два типи багатозадачності: засновану на процесах (process - based) і засновану на потоках (thread - based).

Відмінності багатозадачності на основі *процесів* і *потоків* зводиться до наступного: багатозадачність на основі *процесів* організовується для паралельного виконання програм, а багатозадачність на основі *потоків* - для паралельного виконання окремих частин однієї програми.

Потоки в процесі розділяють спільно використовувані дані і мають власні стеки викликів і локальну пам'ять потоку (Thread Local Storage -TLS).

Головне – це те, що всі потоки виконуються в рамках **спільного адресного простору (в рамках одного процесу**).

Взаємодія між потоками простіше, якщо ці потоки виконуються в рамках одного процесу. При цьому все одно потрібний спеціальний інструментарій для синхронізації потоків: *критичні секції, м'ютекси і семафори*, і залишається багато проблем, вирішення яких вимагає уваги з боку програміста.

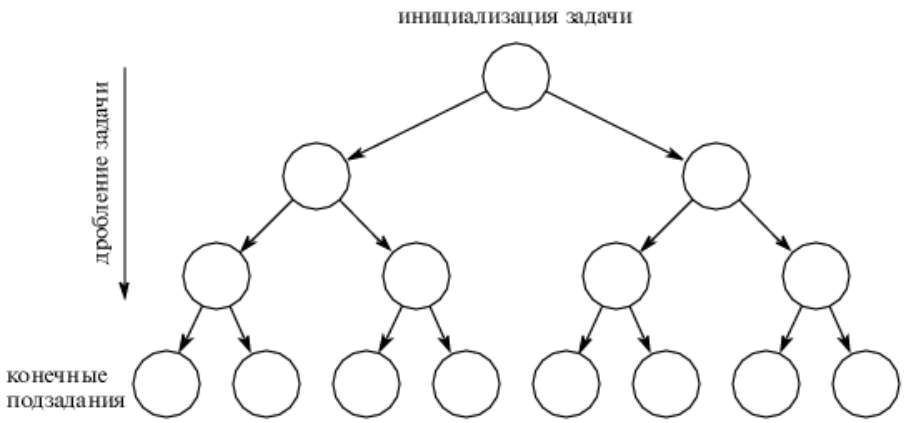
Для забезпечення взаємодії потоків у *різних процесах* використовуються канали, обмін інформацією через які забезпечується спеціальними системними засобами. Загальне управління виконанням потоків здійснюється на рівні ОС.

6. Загальна схема розпаралелювання задач

Для внесення в програму паралелізму необхідно (на етапі проектування) розбити її на незалежні підзадачі, які будуть виконуватися одночасно. Таке розбиття ще називають ***декомпозицією*** задачі.

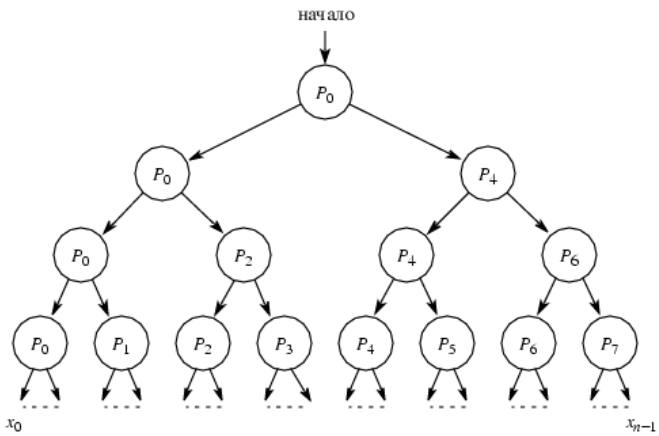
Розпаралелити задачу можна не єдиним способом. Схему розпаралелювання в загальному вигляді зручно графічно зобразити в вигляді розгалужених дерев.

Перший етап: Розбиття задачі на незалежні підзадачі.



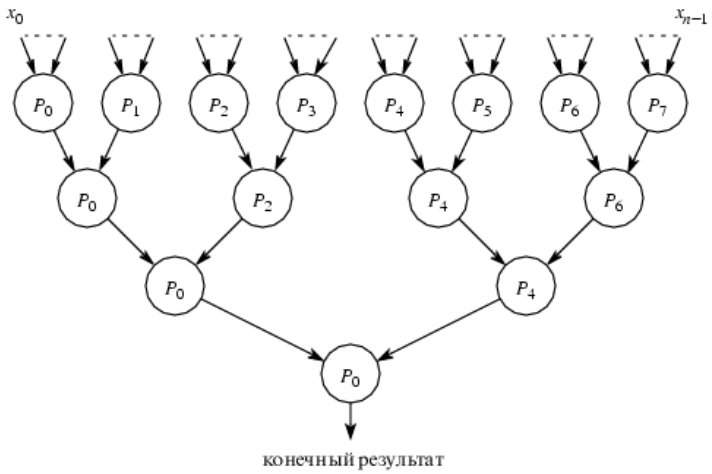
Вычислительное дерево

Другий етап: Призначення конкретних процесорів для виконання кожної підзадачі.



Распределение подзаданий по конкретным процессорам

Третій етап: Збирання результатів роботи окремих процесорів.



Сборка задачи

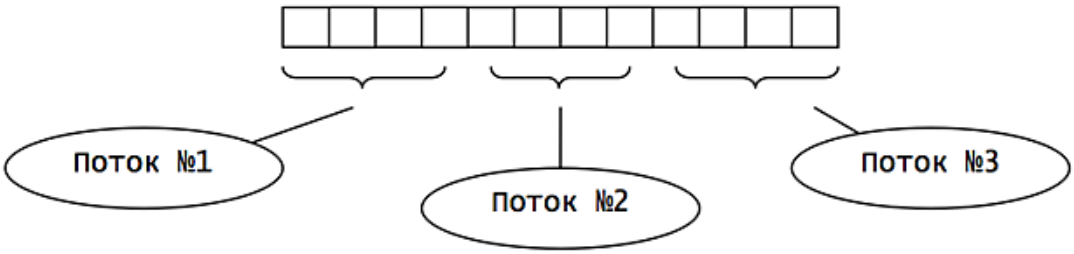
Декомпозиція (розбиття) задачі може бути виконано декількома способами: за задачами, за даними, за інформаційними потоками.

Декомпозиція за задачами (функціональна декомпозиція) припускає призначення різним потокам різних функцій.

Декомпозиція за інформаційними потоками виділяє підзадачі, що працюють з одним типом даних.

Декомпозиція за даними. При декомпозиції за даними кожна підзадача виконує одні і ті самі дії, але з різними даними, наприклад, обробку великого масиву (колекції) даних.

Є два основні принципи розподілу даних між підзадачами - *статичний* і *динамічний*. При статичній декомпозиції фрагменти даних призначаються потокам до початку обробки і, як правило, містять однакову кількість елементів для кожного потоку. Наприклад, розподіл масиву елементів може здійснюватися по рівних діапазонах індексу між потоками (range partition).



Основною перевагою статичного розподілу є незалежність роботи потоків і, як наслідок, немає необхідності в засобах синхронізації для доступу до елементів. Ефективність статичної декомпозиції знижується при різній обчислювальній складності обробки елементів даних. Наприклад, обчислювальне навантаження обробки і-го елемента у масиві може залежати від індексу елементу.

При динамічній декомпозиції кожен потік, що бере участь в обробці, звертається за блоком даних (порцією). Після обробки блоку даних потік звертається за наступною порцією. Динамічна декомпозиція вимагає синхронізації доступу потоків до структури даних. Розмір блоку визначає частоту звернень потоків до структури. Деякі алгоритми динамічної декомпозиції збільшують розмір блоку в процесі обробки. Якщо потік швидко обробляє елементи, то розмір блоку для нього збільшується.

7. Масштабування підзадач

Властивість *масштабованості* полягає в ефективному використанні усіх наявних обчислювальних ресурсів. Паралельне застосування має бути готове до того, що завтра воно запускатиметься на системі з більшими обчислювальними можливостями. Властивість масштабованості тісно пов'язана з вибраним алгоритмом рішення задачі. Один алгоритм дуже добре розпаралелюється, але тільки на дві підзадачі, інший алгоритм дозволяє виділити довільне число підзадач.

Обов'язковою умовою масштабованості програми є можливість параметризації алгоритму залежно від числа процесорів в системі і залежно від поточної завантаженості обчислювальної системи. Така параметризація

дозволяє змінювати число підзадач, що виділяються, за конкретних умов виконання алгоритму. Сучасні платформи паралельного програмування надають засоби для автоматичного балансування навантаження (пул потоків).

Висновки

Під паралельними обчисленнями розуміються процеси обробки даних, в яких одночасно можуть виконуватися декілька операцій комп'ютерної системи. Для того, щоб реалізувати паралельні обчислення, необхідні апаратні і програмні рішення. Всі сучасні процесори і операційні системи підтримують паралельні обчислення. Особливо важливу роль ці обчислення відіграють при вирішенні складних задач, які потребують великих обчислювальних ресурсів.

Контрольні питання і завдання для самостійної роботи

1. У чому полягає різниця між паралельними і розподіленими обчисленнями?
2. В яких задачах доцільно використовувати паралельні обчислення
3. Які є види паралелізму на рівні програмного забезпечення
4. Класифікація Флінна обчислювальних систем
5. Різниця між мультипроцесорами і мультикомп’ютерами
6. Основні етапи розпаралелювання задачі
7. Класифікація Джонсона видів паралелізму
8. Види декомпозиції задач.
9. Які переваги роботи з потоками?
10. У чому полягають основні проблеми організації багатопотокової обробки?

Література

1. Туральчук К. Параллельное программирование с помощью C#. [Електронний ресурс] Режим доступу:
http://www.intuit.ru/studies/courses/5938/1074/info
2. Параллельное программирование для многоядерных процессоров http://www.intuit.ru/department/supercomputing/ppmcp/
3. Параллельное программирование
http://www.intuit.ru/studies/courses/1110/153/info
4. Параллельное программирование с помощью языка C#
http://www.intuit.ru/department/supercomputing/parallcsharp/
5. Эхтер Ш., Роберте Дж. Многоядерное программирование. — СПб.: Питер, 2010. — 316 с.