МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХЕРСОНСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Факультет комп’ютерних наук, фізики та математики

Кафедра комп’ютерних наук та програмної інженерії

Багатопроцесорність: застосування, інструменти і реалізація

Кваліфікаційна робота (проєкт)
на здобуття ступеня вищої освіти «бакалавр»

Виконав: студент 4 курсу 12-441 групи
Спеціальності 121 Інженерія програмного забезпечення
Освітньо-професійної програми «Інженерія програмного забезпечення»
Марисюк Станіслав Олександрович

Науковий керівник: Вейцбліт Олександр Йосипович, доцент кафедри комп’ютерних наук та програмної інженерії.

Рецензент: Захарченко Раїса Миколаївна кандидатка технічних наук, доцентка, доцентка кафедри програмних засобів і технологій Херсонського національного технічного університету.

Івано-Франківськ — 2024

Зміст

Вступ

Важливість багатопроцесорної обробки в сучасному житті полягає в тому, що вона дозволяє значно підвищити продуктивність комп'ютерних систем. Це особливо важливо у великих обчислювальних центрах, у наукових дослідженнях, а також при розробці та використанні складних програмних продуктів.

Це дозволяє створювати і використовувати більш ефективні та енергоефективні комп'ютерні системи в багатьох сферах життя, від персональних пристроїв до промислового і наукового обладнання.

Оскільки вони забезпечують високу продуктивність та ефективність обчислень. Це особливо важливо для завдань які потребують великої потужності для обчиленя та обробки великої кількості даних.

У виробництві та автоматизації багатопроцесорні системи використовуються для керування складними технічними процесами.

У наукових дослідженнях багатопроцесорність дозволяє проводити складні моделювання та симуляції, які неможливі за допомогою одного процесора.

Враховуючи ці фактори, мультипроцесорність є однією з найважливіших тенденцій розвитку сучасних комп'ютерних технологій і буде продовжувати впливати на багато сфер життя.

Таким чином, мультипроцесорність є ключовим елементом розвитку сучасних комп'ютерних технологій і відкриває нові можливості для прогресу в різних галузях.

Розділ 1
Багатопроцесорність

Багатопроцесорність, це використання двох або більше фізичних процесорів в одній комп'ютерній системі. Це дозволяє системі ефективний розвиток, розвиток їх між процесорами. Операційна система та багатопроцесорна програма про те, що можна зробити з процесорами.

Ось список типів багатопроцесорності:

Симетрична багатопроцесорна обробка (SMP): усі процесори в системі є рівноправними та можуть створювати нові завдання.

Симетрична багатопроцесорна обробка (SMP) - це багатопроцесорна комп'ютерна архітектура, в якій два або більше однакових процесорів спільно використовують пам'ять. Більшість сучасних багатопроцесорних систем використовують архітектуру SMP.

У SMP-системах будь-яке завдання може виконуватися на будь-якому процесорі, незалежно від того, де в пам'яті зберігаються дані завдання. За належної підтримки операційної системи SMP-системи можуть легко переміщувати завдання між процесорами та ефективно розподіляти навантаження.

Однак, оскільки пам'ять працює набагато повільніше, ніж процесори, які до неї звертаються, навіть однопроцесорним комп'ютерам доводиться витрачати значний час на отримання даних з пам'яті; в SMP-системах тільки один процесор може одночасно звертатися до пам'яті, що може призвести до складних проблем.

Асиметрична багатопроцесорна обробка (ASMP): кілька процесорів можна зарезервувати для спеціальних цілей.

(ASMP) - тип мультипроцесорної обробки, що використовувався до створення технології симетричної мультипроцесорної обробки (SMP). В асиметричних архітектурах відрізняються як характеристики процесорів (продуктивність, надійність, система команд і т.д. аж до моделі мікропроцесора), так і функціональні ролі, які їм відводяться в системі.

Наприклад, деякі процесори призначені для роботи в якості хостів, інші - для керування підсистемами вводу/виводу, а треті - для інших спеціалізованих цілей. Функціональна неоднорідність в асиметричних архітектурах призводить до структурних відмінностей в частинах системи, які містять різні процесори в системі.

Асиметрична багатопроцесорна обробка (ASMP) - це технологія, яка дозволяє багатоядерним процесорам працювати разом більш ефективно; ASMP збільшує обчислювальну потужність всієї системи, дозволяючи кожному ядру виконувати різні завдання.

Багатопроцесорна обробка з нерівномірним доступом до пам’яті (NUMA): системні ресурси можуть бути спільними між процесорами, які мають різний доступ до пам’яті.

Нерівномірний доступ до пам'яті (NUMA) - це схема комп'ютерної пам'яті, що використовується в багатопроцесорних системах. У NUMA час доступу до пам'яті залежить від розташування пам'яті відносно процесорів. Згідно з NUMA, локальна пам'ять процесора (локальна пам'ять інших процесорів або пам'ять, що розділяється процесорами) є швидшою, ніж зовнішня пам'ять

(локальна пам'ять інших процесорів або пам'ять, що розділяється процесорами).

NUMA додає кілька проміжних рівнів пам'яті, які спільно використовуються мікропроцесорами, так що не всі операції доступу до даних відбуваються на головній шині; NUMA можна розглядати як "кластер в коробці".

Коли процесор шукає дані за заданою адресою пам'яті, він спочатку шукає їх у кеші L1 власного мікропроцесора, потім у більших кешах L1 і L2, потім у кеші третього рівня, передбаченому конфігурацією NUMA, а потім у "віддаленій пам'яті", розташованій поруч з іншими процесорами, де шукаються дані.

Також важливо розуміти багатопотоковість, яка підрозділяється на багатопроцесорність. Багатопотоковість дозволяє процесу складатися з кількох потоків, які виконуються паралельно в одному адресному просторі.

Багатопроцесорність дозволяє робити різноманітні операції, які можуть значно підвищити продуктивність системи.

Паралельні обчислення: обробляє великі обсяги даних швидше, використовуючи кілька процесорів для одночасного виконання обчислень.

Розподіл завдань: Завдання автоматично розподіляються між процесорами, щоб збалансувати робочі навантаження і забезпечити більш ефективну роботу.

Висока надійність: якщо один процесор виходить з ладу, інші продовжують виконувати завдання, гарантуючи безперервність роботи системи.

Масштабованість: додаткові процесори можуть бути додані для збільшення продуктивності системи без зміни програмного забезпечення.

Спеціалізована обробка: деякі процесори можуть бути оптимізовані для виконання спеціальних завдань, таких як обробка графіки або шифрування даних.

Ці операції можуть застосовуватися в різних сферах-від наукових досліджень до великих комерційних центрів обробки даних.

Багатопотоковий підхід забезпечує багато переваг для додатків, особливо з точки зору ефективності та швидкості виконання. Основними причинами, чому багатопоточність приносить користь додаткам, є

Паралельне виконання: багатопоточність дозволяє програмам виконувати декілька завдань одночасно, що значно скорочує час виконання.
Ефективне використання ресурсів: програми можуть краще використовувати ресурси процесора, розподіляючи обчислення між різними потоками.
Зворотний зв'язок з користувачем: Для додатків з графічним інтерфейсом багатопоточність гарантує, що інтерфейс залишається чуйним навіть при виконанні складних обчислень.
Зменшення затримок: потоки можуть виконувати операції вводу/виводу паралельно, таким чином зменшуючи затримки.

Спрощення програмування: в деяких випадках багатопоточність спрощує структуру програми та полегшує виконання складних завдань.
Використання багатопоточності вимагає від програміста розуміння концепції синхронізації та уникнення проблеми конфліктного доступу до спільних ресурсів. Однак, якщо багатопоточність використовується правильно, вона може значно підвищити продуктивність та ефективність додатків.

Синхронізація даних

Синхронізація даних - це процес забезпечення узгодженості даних на всіх пристроях, незалежно від того, на якому з них відбулися зміни.Іншими словами,мова йде про усунення відмінностей між двома копіями даних. Передбачається, щоці копії раніше були ідентичними,а потім одна або обидві змінилися незалежно один від одного.

Синхронізація даних використовується в різних галузях,зокрема в інформатиці та фізиці.Вона гарантує,що цілісність файлів,які використовуються на різних пристроях,завжди підтримується.Цей механізм відстежує всі зміни і дозволяє зручно обмінюватися даними між пристроями на одній або різних платформах,системах.

Синхронізація даних особливо корисна,коли дані зберігаються або обробляються на декількох пристроях, і всі ці пристрої повинні мати доступ до останньої версії даних. Це стосується,наприклад,роботи з документами в хмарних сервісах,синхронізації контактів і календарів між комп'ютерами та мобільними пристроями або забезпечення узгодженості даних урозподіленій базі даних.
Програми можуть уникнути проблем із синхронізацією, використовуючи різні методи та інструменти. Деякі з них наведено нижче:

Використання мьютексів та семафорів: Це базові інструменти синхронізації, які контролюють доступ до спільних ресурсів і запобігають одночасному доступу до них декількох потоків.

Умовні змінні: потоки можуть бути заблоковані до виконання певних умов, що допомагає уникнути нестачі ресурсів.

Блокування на рівні об'єктів: об'єктно-орієнтовані мови програмування часто пропонують можливість блокувати окремі об'єкти, надаючи до них синхронізований доступ.

Атомарні операції: завдяки використанню атомарних операцій над змінними, операції читання та запису можна виконувати без переривання.

Монітори: високорівневі механізми синхронізації, включаючи м'ютекси та змінні стану, які контролюють доступ до об'єктів.

Бар'єр: синхронізує потоки на певному етапі виконання і гарантує, що всі потоки досягнуть цього етапу перед продовженням.

Транзакційна пам'ять: система, яка дозволяє потокам атомарно виконувати послідовність операцій з пам'яттю.

Вона використовує такі патерни проектування, як виробник-споживач і читач-записувач.

Використовуйте фреймворки та бібліотеки, які надають вбудовані інструменти синхронізації, такі як Synchronous Suite від Java та asyncio від Python.

Комплексне тестування: проводьте стрес-тести та використовуйте інструменти, які виявляють нестачу ресурсів та інші проблеми синхронізації. Слід зазначити, що неправильне використання інструментів синхронізації може призвести до таких проблем, як тупики та накладні витрати. Тому розробники повинні ретельно вибирати метод синхронізації відповідно до конкретних вимог програми.

Розділ 2

Конструкції

Розподіл навантаження у двовимірних конструкціях є важливим фактором при проектуванні та аналізі міцності і стійкості різних об'єктів.

Навантаження можуть бути розподілені в різних напрямках, таких як довжина, ширина і площа поверхні.

При проектуванні конструкцій необхідно враховувати різні типи навантажень, такі як власна вага елементів, тиск газів і рідин, розподілена вага сипучих матеріалів і вітрові навантаження. Ці навантаження можуть бути рівномірно розподіленими, лінійними, трикутними, трапецієподібними і нелінійними.

Для аналізу і розрахунку міцності і стійкості конструкцій використовуються методи теорії пружності і опору матеріалів. Ці методи дозволяють визначити величину і розташування еквівалентних навантажень та оцінити деформації і напруження елементів конструкцій.

При розрахунку і проектуванні двовимірних конструкцій важливо враховувати кожен вид навантаження і підбирати оптимальну форму і розміри елементів для забезпечення надійності і довговічності об'єкта.

Важливість аналізу розподілу навантаження двовимірних конструкцій обумовлена необхідністю забезпечення безпеки і довговічності будівель і споруд. Аналіз допомагає виявити слабкі місця в конструкції, запобігти можливим аваріям та оптимізувати використання ресурсів.

Новітні будівельні норми і правила вимагають ретельного аналізу розподілу навантажень в 2D конструкціях, оскільки це впливає на міцність, стійкість і довговічність будівель. Неправильні розрахунки навантажень можуть призвести до деформації конструкції, руйнування та втрати несучої здатності.

Аналіз розподілу навантаження передбачає визначення опорних реакцій, моментів і зусиль в елементах конструкції. Це дозволяє оцінити роботу навантаженої конструкції та виявити зони концентрації напружень.

Використовуючи сучасні програмні пакети і методи розрахунку, розподіл навантаження в 2D конструкціях можна проаналізувати швидко і точно. Однак для отримання достовірних результатів необхідно враховувати всі фактори, що впливають на роботу конструкції, включаючи геометричні параметри, матеріали, з'єднання і навантаження.

Тому важливість аналізу розподілу навантажень 2D конструкцій зумовлена необхідністю забезпечення безпеки та довговічності будівель і споруд. Такий аналіз допомагає виявити слабкі місця в конструкціях, запобігти можливим аваріям та оптимізувати використання ресурсів.

Обчислення навантаження

Приклад обчислення навантаження на 2D-конструкцію:

Скажімо, у нас є балка довжиною 5 метрів і шириною 10 сантиметрів. Балка виготовлена зі сталі з щільністю 7850 кг/м3 і модулем пружності 200 ГПа. Навантаження на балку становить 2 тонни.

Розрахуємо площа поперечного перерізу балки:

Площа = Ширина * довжина = 10 см * 5 м = 0,05 м2

Розрахуємо момент інерції балки:

Момент інерції = Площа * відстань від центру ваги до краю = 0,05 м2 * (5 м / 12 ² 2 = 0,0125 м

Найефективніші конструкції до опору навантажень

У сучасному будівництві однією з головних завдань є створення ефективних і надійних конструкцій, здатних протистояти різним навантаженням. У цій статті ми розглянемо основні критерії, які визначають ефективність таких конструкцій.

Критерії ефективності

Вартість.

- Важливим критерієм є вартість конструкції. Вона повинна бути мінімальною в порівнянні з іншими можливими рішеннями. Однак на практиці цей критерій часто буває недостатнім, так як необхідно враховувати і інші фактори.

Довговічність.

- Конструкція повинна бути довговічною і витримувати різні впливи протягом усього терміну служби. Це забезпечує стабільність і безпеку будівлі або споруди.

Шкода навколишньому середовищу.

- При виборі конструкції слід враховувати вплив на навколишнє середовище. Наприклад, використання екологічно чистих матеріалів або застосування енергоефективних технологій може зменшити негативний вплив на природу.

Втрати і збитки.

- Необхідно оцінювати можливі втрати і збитки, пов'язані з експлуатацією об'єкта. Це може включати витрати на ремонт, технічне обслуговування та заміну елементів конструкції.
-

Вплив на інфраструктуру.

- Важливо враховувати вплив об'єкта на інфраструктуру району або регіону. Наприклад, будівництво висотних будівель може зажадати додаткових заходів щодо забезпечення стійкості інфраструктури.

Розрахунок навантажень на 2D конструкцію

Розрахунок навантажень на 2D конструкції включає кілька основних етапів:

1. Упорядкування розрахункової схеми.
2. Визначення та додаток навантажень.
3. Визначення зусиль та деформацій у розрахунковій схемі.
4. Виконує перевірочний розрахунок конструкцій на задані вимоги.

Ось приклад розрахунку навантаження на стіни:

Визначаємо площу стін S = AxB, де S - площа, A - ширина, B - висота.
Визначаємо об'єм стінок V=SxT, де V - об'єм, S - площа, T - товщина стінок.
Визначаємо вагу стін Q = Vxg, де Q - вага, V - об'єм, g - питома вага матеріалу стіни.
Визначаємо питому навантаження, з якого стіни будівлі тиснуть на фундамент (кг/м2) q=Q/s, де s - площа спирання несучих конструкцій на фундамент.
Для обчислення навантаження у точці на двовимірній балці використовується формула:
F(x,y)=q∗ (x^2 +y^2),
де F(x, y) - навантаження в точці з координатами (x, y), а q - інтенсивність навантаження.
M=q∗ l/8,
де M - згинальний момент, q - рівномірно розподілене навантаження, l - Довжина балки.

Ось докладніші кроки:

1. Розділіть балку на дві ділянки.Це можна зробити, наприклад, посередині балки.

2. Врахуйте розподілене навантаження на кожній ділянці балки як тиск, що дорівнює q1/4, прикладений докінцівкожноїділянки балки.

3. Об'єднайте дві сили у середині балки. Це дасть вам статично еквівалентне навантаження.

4. Це еквівалентне навантаження буде додане в середині балки,щодастьвамточне уявлення про вплив навантаження на балку.

Термінологія, пов'язана з обчисленням навантаження на конструкції:

1. зосереджена сила (concentrated load) - це сила, прикладена в одній точці;
2. суцільне навантаження (distributed load) - це навантаження, точки докладання якого безперервно заповнюють даний відрізок або майданчик;
3. інтенсивність навантаження (intensity of distributed load) - це межа відношення величини рівнодіючого навантаження до величини площі, якщо остання прагне нуля;
4. рівномірно розподілене навантаження (continuous load) — це навантаження, постійна інтенсивність якого посідає одиницю довжини чи площі поверхні;
5. постійне навантаження (dead load) - це навантаження, що діє незмінно за величиною та напрямом;
6. тимчасове навантаження (live load) - це навантаження, яке може діяти або бути відсутнім залежно від її значення для елемента, що розраховується;
7. рухоме навантаження (moving load) - це навантаження, яке може займати різне становище на системі;

8. статична навантаження (statical load) - це навантаження, положення, напрям і інтенсивність якої приймаються при розрахунку не залежними від часу;
9. динамічне навантаження (dynamic load) - це навантаження, зміна величини, напряму або положення якої відбувається настільки швидко, що при розрахунку споруди необхідно враховувати інерційні сили;
10. повторно змінне навантаження (repeated-fluctuating load) - це навантаження, що багаторазово змінює з часом значення або значення та напрямок;
11. невигідне розрахункове поєднання навантажень (unfavourable load combination) - це сукупність постійних і тимчасових навантажень, що відповідає максимальному позитивному або найбільшому за абсолютною величиною негативного значення обчислюваної величини;
12. стежить навантаження (follower load) - це навантаження, напрям якої залежить від деформації системи, що навантажується;
13. несуча здатність споруди (ultimate load carrying capacity) - це характеристика споруди, яка виражається величиною навантаження, що відповідає граничному стану споруди за міцністю;
14. критичне навантаження (critical load) - це найменше навантаження, при якій відбувається втрата стійкості системи.

Стратегії балансування навантаження

У багатопроцесорних системах для ефективного розподілу завдань між процесорами використовуються різні стратегії балансування навантаження. Деякі з них представлені нижче:

1. **Стратегія граничного значення**: Ця стратегія визначає межу навантаження для кожного процесора. Якщо навантаження на процесор перевищує цю межу, завдання перерозподіляється.

2. **Жадібна стратегія**: в цій стратегії вхідне завдання призначається процесору незалежно від поточного завантаження процесора.

3. **Найкоротша стратегія**: завдання розподіляються між процесорами відповідно до їх поточного завантаження, при цьому пріоритет надається процесору з найменшим завантаженням.

Ці стратегії можуть бути статичними або динамічними. Статичні стратегії заздалегідь визначають розподіл завдань, тоді як динамічні стратегії адаптують розподіл завдань у реальному часі у відповідь на зміни робочого навантаження.

Стратегія граничного значення використовується, щоб визначити, коли навантаження на процесор досягає певного рівня і завдання потрібно перерозподілити між процесорами. Це запобігає перевантаженню окремих процесорів і дозволяє більш ефективно використовувати системні ресурси. Основні аспекти стратегії порогових значень наступні

Встановлення порогу: для кожного процесора встановлюється максимально допустиме навантаження.

Моніторинг: система безперервно відстежує завантаження кожного процесора.

Реактивна: якщо навантаження на процесор перевищує встановлений поріг, система перерозподіляє завдання, щоб зменшити навантаження.
Адаптивна: система може адаптуватися до змін робочого навантаження, динамічно змінюючи порогові значення.

Ця стратегія може бути статичною або динамічною. У статичному випадку пороги встановлюються один раз і не змінюються під час роботи системи. У динамічному варіанті пороги можуть змінюватися відповідно до поточного стану та навантаження системи.

Жадібна стратегія - це підхід, який розподіляє завдання між процесорами на основі найшвидшого доступного варіанту без урахування загальної оптимальності розподілу. Це означає, що кожне нове завдання призначається процесору, який може виконати його найшвидше на даний момент, незалежно від того, як це вплине на інші завдання або загальну продуктивність системи.

Деякі ключові особливості жадібної стратегії перераховані нижче:

Простота реалізації: Жадібні алгоритми, як правило, легше реалізувати, ніж інші стратегії балансування навантаження.
Час відгуку: Жадібні стратегії не вимагають складного аналізу стану системи і можуть швидко розподіляти завдання.
Потенціал перевантаження: Жадібні підходи можуть призвести до нерівномірного розподілу навантаження, коли деякі процесори будуть перевантажені, а інші - недовантажені.
Відсутність глобальної оптимізації: жадібні стратегії не враховують загальне навантаження системи, і тому розподіл ресурсів може бути неоптимальним.

Жадібні стратегії ефективні в системах з невеликою кількістю завдань або там, де складність завдань і час їх виконання є порівнянними. Однак більш складні та динамічні системи з різною складністю завдань і часом виконання можуть вимагати більш інтелектуального підходу для ефективного балансування навантаження.

Найкоротша стратегія або стратегія найкоротшого часу (Shortest Job Next, SJN), - це метод балансування навантаження в багатопроцесорних системах: Стратегія SJN вибирає завдання з найкоротшим очікуваним часом завершення або з найменшим обсягом роботи, що залишився.

Основні характеристики цієї стратегії наступні

Пріоритет часу: завданням з найкоротшим часом виконання надається найвищий пріоритет.
Ефективність: Стратегія спрямована на мінімізацію часу очікування для всіх процесів.
Прогнозування: необхідно вміти прогнозувати час виконання завдань.
Динамічність: Залежно від часу виконання, нові завдання можуть бути додані до черги виконання, таким чином адаптуючись до змін у робочому навантаженні.
Ця стратегія ефективна для скорочення середнього часу очікування і збільшення пропускної здатності системи, але вона може призвести до проблеми "голодування", коли довгострокові завдання постійно стикаються з короткостроковими, які можуть затримуватися на невизначений час.
Ось ще деякі види стратегії балансування навантаження наведені нижче

Круговий алгоритм: запити розподіляються між серверами в певному порядку.

Сервер з найменшим навантаженням: завдання розподіляються на сервер з найменшим поточним навантаженням.

Сервер з найкращим часом відгуку: завдання розподіляються на сервер з найкращим часом відгуку.

Випадковий розподіл: завдання розподіляються випадковим чином між доступними серверами.

Розділ 3

Функціонал програми

Функціонал програми полягає в тому що вона шукає найефективнішу конструкцію при заданій Користувачем навантаженні .

Конструкцію в програмі можна описати наступним чином є 2 паралельні пластини N довжини з'єднані між собою балками які можуть з'єднувати їх перебуваючи під кутом , у балок є опір до навантаження h при перевищенні цього порогового значення балка валитися.

На конструкцію в програмі буде проводиться навантаження на верхню пластину в точці з силою U, користувач має можливість змінювати такі параметри як розмір пластин і їх відстань між ними , силу тиску і точку куди буде додаватися тиск.

Після введення користувачем всіх даних програма починає свою роботу і шукає конструкцію з най меншим числом балок здатну витримати тиск в заданій тачці робить вона це наступним шукає для кожної випадково згенерованого числа балок з випадковими точками зіткнень балок і пластин але у балок не може бути точки дотику верхньої і нижньої пластини повністю ідентичними тільки в одній точці вони можуть з'єднаються з порожнинної далі шукається для кожної балки точка в якій балка здатна витримати максимальне навантаження далі шукається навантаження після цього обчислюється внесок кожної балки для погашення тиску якщо конструкція не справляється з навантаженням і не досягнуто максимально можливе числа балок додається ще одна балка в конструкцію в інших випадках зберігається поточна конструкція для подальшого пошуку ефективної конструкції при випадки коли конструкція справляється з навантаженням забирається 1 балка при не досягнуто максимально можливе числа балок цей пункт пропускається далі 1 або 3 балки змінюють свої точки зіткнень випадковим

чином і далі для них максимальне навантаження оновлює їх внесок для погашення навантаження і все зациклюється аж до знаходження конструкції здатної витримати навантаження з най менших витрат ресурсів.

Взаємодію з багатопроцесорністю

Далі ми розглянемо як в цій програмі організовано взаємодію з багатопроцесорністю – для того щоб була можливість використовувати багатопроцесорність в c++ використовується бібліотека thread в C++ надає функціонал для роботи з потоками . Ось деякі ключові особливості та методи, які надає ця бібліотека:

1. **Конструктор:**

- thread (): створює новий об'єкт thread, який не пов'язаний з потоком виконання.
- thread (викликаний об'єкт): створює новий потік виконання та викликає викликаний об'єкт у цьому потоці.

2. **Метод:**

- join (): блокує виклик потоку, поки потік, пов'язаний з об'єктом thread, не завершить його виконання.
- detach (): Від'єднує потік від об'єкта thread, роблячи операційну систему відповідальною за звільнення ресурсів потоку.
- joinable (): перевіряє, чи можливо приєднання (join) потоку.
- get_id (): повертає унікальний ідентифікатор пов'язаного потоку.
- hardware_concurrency (): повертає приблизну кількість контекстів апаратного потоку, доступних для виконання.

3. **Оператор:**

- operator= (): Пов'язує потік з поточним об'єктом thread.

Об'єкти thread можна переміщати, але не копіювати, що гарантує, що кожен потік виконання може бути пов'язаний лише з одним об'єктом thread. Кожен потік виконання має унікальний ідентифікатор типу thread:: id1.

Таким чином за допомогою бібліотеки thread і використанні декількох потоків і виконується паралельні обчислення і потрібні вони для економії часу при обчисленні таких завдань як пошук точки в якій балка здатна витримати максимальне навантаження і пошуку відстані для цих точок відстань до точки навантаження.

Висновок

Багатопроцесорна обробка є фундаментальним аспектом сучасних комп'ютерних систем і відіграє важливу роль у підвищенні продуктивності та ефективності обчислень. Вона забезпечує паралельну обробку даних і відіграє важливу роль у широкому спектрі застосувань, від персональних пристроїв до промислових і наукових досліджень. Завдяки своїй масштабності та можливостям паралельного виконання, багатопроцесорні системи пропонують нові можливості для технологічного розвитку, особливо в галузі штучного інтелекту та машинного навчання.

У майбутньому, з розвитком технологій і зростанням попиту на обчислювальні потужності, багатопроцесорність залишатиметься важливим елементом в дизайні та архітектурі комп'ютерних систем, сприяючи прогресу в різних сферах людської діяльності. Тому багатопроцесорна обробка є не тільки актуальною, але й перспективною технологією, яка продовжуватиме впливати на розвиток сучасного світу.

Джерела

1. ASMP vs. SMP (ipv7.net)
2. Full Introduction to NUMA (Non-Uniform Memory Access) - MiniTool
3. Non Uniform Memory Access (NUMA) Explained (ecomputertips.com)
4. Балансировка нагрузки: основные алгоритмы и методы / Хабр (habr.com)
5. Балансировщик нагрузки с Docker nginx. Сравнение стратегий работы (badtry.net)
6. ОБЗОР МЕТОДОВ БАЛАНСИРОВКИ НАГРУЗКИ В ГЕТЕРОГЕННЫХ РАСПРЕДЕЛЕННЫХ ФАЙЛОВЫХ СИСТЕМАХ - Фундаментальные исследования (научный журнал) (fundamental-research.ru)
7. Класс thread | Microsoft Learn
8. Жадный алгоритм на примере: что такое, метод и подход (guru99.com)
9. Жадные алгоритмы / Хабр (habr.com)