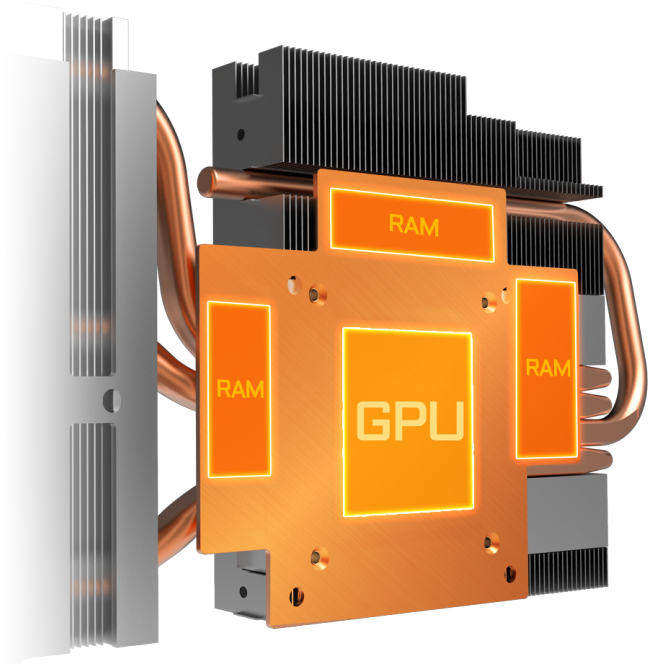
МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

Гетерогенні паралельні обчислення

9 лютого 2024 р.



МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

Кочура Ю. П.

Гетерогенні паралельні обчислення

Навчальний посібник

Рекомендовано для здобувачів ступеня “магістр”, які навчаються за освітніми програмами «Комп’ютерні системи та мережі» спеціальність 123 «Комп’ютерна інженерія» та «Інженерія програмного забезпечення комп’ютерних систем» спеціальність 121 «Інженерія програмного забезпечення»

Електронне видання

Навчальний посібник перебуває на етапі написання. Остання доступна версія цього документа за QR-кодом:

Укладач буде дуже вдячний за надану допомогу та пропозиції щодо наповнення та покращення цього посібника. Повідомлення про проблеми надсилайте через github: https://github.com/hpc-book/hpc/issues

Чернетка. Версія 0.1.0

Зміст

Розділ 1

Вступ до гетерогенних паралельних обчислень

“Я не маю ніякого таланту, тільки допитливість.
Відповідно, віднадає питання про спадковість.”

– Альберт Ейнштейн

Гетерогенні обчислення відносяться до систем, які використовують більш ніж один тип процесора або ядра. Ці системи підвищують продуктивність або енергоефективність не лише за рахунок додавання процесорів того самого типу, а й за рахунок додавання різних співпроцесорів, які зазвичай включають спеціалізовані можливості обробки для виконання певних завдань.

У традиційних гомогенних комп'ютерних системах центральні процесори (CPU) є основним інструментом обчислень, що відповідає за виконання завдань загального призначення. Однак, їх здатність ефективно обробляти різноманітні паралельні навантаження часто обмежена. З іншого боку, графічні процесори (GPU), які першочергово були розроблені для візуалізації графіки, чудово справляються з паралельними обчисленнями завдяки великій кількості ядер і спеціалізованій архітектурі. Гетерогенна паралельна система об'єднує CPU та GPU в одну систему. Центральний процесор залишається відповідальним за керування загальними системними завданнями, а графічний процесор використовується для інтенсивних обчислювальних паралельних операцій. Такий підхід дозволяє одночасно виконувати різні частини програми на визначеному спеціалізованому процесорі, покращуючи таким чином загальну продуктивність й скорочуючи час виконання.

Одним із ключових аспектів гетерогенних паралельних обчислень є потреба в спеціалізованих моделях і структурах програмування. Традиційні послідовні мови програмування погано підходять для використання паралелізму, який пропонують неоднорідні архітектури. Тому розробники повинні використовувати моделі програмування, щоб ефективно розподіляти обчислювальні завдання між кількома процесорами. Окрім цього, ще один важливий фактор у такого типу обчислень є переміщення даних. Оскільки процесори та графічні процесори мають окремі просторої пам'яті, передача даних між цими пристроями може призвести до значних витрат. Оптимізація переміщення даних і мінімізація передачі даних є критично важливими для досягнення ефективного паралельного виконання.

1.1 Роль гетерогенних паралельних обчислень

Гетерогенні паралельні обчислення відіграють важливу роль для інженерів та дослідників, надаючи свої переваги та вирішуючи проблеми масштабованості сучасних обчислень, які пов'язані зі зростанням попиту на обчислювальну потужність. Оскільки обсяг даних і складність обчислень продовжують зростати, паралельні обчислення стають необхідними для більш ефективного вирішення поставлених задач.

Гетерогенні системи активно використовуються у таких сферах: обробка великих даних, машинному навчанні, комп'ютерній графіці, наукових дослідженнях та обчисленнях.

Що стосується науки, то гетерогенні паралельні системи використовуються практично у всіх областях, включаючи обчислювальну фізику, обчислювальну хімію, біоінформатику та геноміку, моделювання клімату, прогнозування погоди, фізику високих енергій, тощо. Гетерогенні системи дозволяють симулювати складні фізичні явища, такі як молекулярна динаміка та динаміка рідини, що дозволяє швидше й точніше моделювати, допомагаючи дослідникам у вивченні складних систем і явищ.

В обчислювальній хімії, для прикладу, гетерогенні системи допомагають науковцям у питання, які пов'язані з відкриттям нових ліків та їх тестуванням. Графічні процесори прискорюють виконання інтенсивних обчислювальних завдань, які виникають під час моделювання молекулярної динаміки та квантово-механічних обчислень, сприяючи швидшому аналізу і у деяких випадках глибшому розумінню хімічних систем.

Гетерогенні системи відіграють важливу роль для моделювання клімату та прогнозуванні погоди, обчислюючи складні числові моделі, що побудовані на кількісних показниках температури, атмосферного тиску, особливостях циклонів, геологічної активності та показників життєдіяльності людини. Можливість виконувати паралельно обчислення на кількох тисячах ядер одночасно дозволяє ефективно здійснювати передбачення атмосферної динаміки, океанічної циркуляції та кліматичних особливостей, що сприяє покращенню прогнозів погоди.

Поруч із фундаментальними науками, ще однією популярною галуззю застосування є обробка великих та надвеликих масивів даних. Аналіз даних є необхідним для пошуку та запровадження ефективних рішень з метою вчасного реагування на нові виклики. У машинному навчанні тренування складних моделей на великих наборах даних потребує виконання значних обчислень. Завдяки використанню можливостей паралельної обробки у гетерогенних системах значно скорочується час навчання моделі, що призводить до швидшої ітерації та підвищення точності моделі, тому гетерогенні паралельні системи мають ключове значення для ефективного моделювання в машинному навчанні. Коли модель навчена, їй потрібно обробити нові дані та зробити прогнози. Графічні процесори або спеціальні прискорювачі штучного інтелекту використовуються для виконання паралельних обчислень із вхідними даними, що забезпечує швидші та точніші прогнози. Це особливо важливо для здійснення прогнозу в режимі реального часу або на периферійних пристроях, де велика затримка є критично негативною.

Ще однією дуже популярною сферою застосування гетерогенних систем є графічний процесинг у відеоіграх та комп'ютерній графіці. Одним з головних завдань тут є забезпечення реалістичного рендерингу графіки у відеоіграх. Графічні процесори, які мають кілька тисяч ядер, відмінно справляються з виконанням складних обчислень, які необхідні для візуалізації 3D-графіки. Завдяки цьому з'являється можливість візуалізації в реальному часі складних сцен з моделюванням освітлення, затінення та ігрової фізики. Також гетерогенні системи відіграють важливу роль у завданнях обробки зображень для віртуальної та доповненої реальність.

1.1.1 Програмні та архітектурні рішення для гетерогенних паралельних обчислень

Гетерогенні паралельні системи використовують спеціалізовані архітектурні рішення для оптимізації продуктивності, що відіграє вирішальну роль у забезпеченні ефективних паралельних обчислень і досягненні високопродуктивних обчислень.

1.1.2 Типи обчислювальних пристроїв

Гетерогенні системи складаються з різнотипних обчислювальних пристроїв, які націлені на ефективне виконання конкретних задач. Першим та найголовнішим з них є центральний процесор (CPU, ЦП). ЦП розроблений для обчислень загального призначення та вирішує такі завдання, як запуск операційних систем, виконання програмних додатків і керування системними операціями в цілому. ЦП оптимізовані для послідовної обробки і мають відносно невелику кількість потужних ядер.

Графічний процесор (GPU, ГП) – це спеціалізовані процесори, що спершу створювались для відтворення графіки, однак згодом перетворилися на потужні паралельні обробники. ГП досить ефективно може виконувати паралельне опрацювання інструкцій за рахунок відносно великої кількості менш потужних ядер порівняно з ЦП.

Ще одним популярним рішенням паралельної обробки даних є програмована вентильна матриця (FPGA). Вона представляє собою програмовані логічні пристрої, які можна переконфігурувати для виконання вузько спеціалізованих спеціальних завдань. На відміну від CPU та GPU, які мають фіксовану архітектуру, FPGA дозволяють розробникам створювати власні цифрові схеми, адаптовані до їхніх програм.

Спеціальна інтегральна схема (ASIC) – це спеціалізовані інтегральні схеми для виконання конкретних завдань. На відміну від FPGA, ASIC мають фіксовану архітектуру та оптимізацію. ASIC забезпечують найвищу продуктивність і енергоефективність для цільового робочого навантаження. Вони зазвичай використовуються в спеціалізованих програмах, таких як мережеве обладнання, штучний інтелект, видобуток криптовалюти та вбудовані системи.

1.1.3 Архітектурні рішення

Гетерогенна системна архітектура (Heterogeneous System Architecture, HSA) – галузева ініціатива, спрямована на створення стандартизованої платформи для гетерогенних обчислень. Вона прагне створити загальну апаратну та програмну екосистему для процесорів, графічних процесорів та інших прискорювачів, що дозволяє їм бездоганно працювати разом. HSA зосереджується на покращенні продуктивності, енергоефективності та можливості програмування в гетерогенних системах. До основних функцій HSA входять:

- Спільна віртуальна пам'ять
- Гетерогенна черга (hQ)
- Гетерогенний уніфікований доступ до пам'яті (hUMA)
- Гетерогенна обчислювальна абстракція (hC)
- HSA Runtime

Уніфікований механізм черги (hQ) дозволяє різним блокам обробки подавати та виконувати завдання незалежно один від одного, завдяки чому можливо планувати й виконувати завдання на певному процесорі, опираючись на наявні ресурси.

hUMA – це архітектура пам'яті, яка забезпечує узгоджений доступ для кількох процесорів, що дозволяє центральному та графічному процесорам узгоджено обмінюватися даними.

Гетерогенна обчислювальна абстракція (hC) надає розробникам можливість писати код, націлений на гетерогенні системи завдяки набору інструкцій та API, які абстрагують базове обладнання.

HSA Runtime — це програмний компонент, який забезпечує необхідну інфраструктуру для виконання програм, керуючи чергою та плануванням завдань на різних процесорах.

1.1.4 Інтерфейси та програмні моделі для гетерогенних систем

Для програмування гетерогенних систем на даний момент розроблено кілька інтерфейсів і моделей програмного забезпечення, щоб полегшити ефективне використання різних процесорів. Вони дозволяють переміщенням розробникам описувати паралелізм, керувати переміщення даних і оркеструвати обчислення в різнорідних архітектурах.

OpenCL (Open Computing Language) – це відкритий стандарт, який забезпечує структуру програмування для різнорідних систем. Він дозволяє створювати програмне забезпечення, яке може виконуватися на різних платформах, включаючи центральні процесори, графічні процесори та FPGA. OpenCL використовує C-подібну мову та надає API для паралелізму завдань, керування пам'яттю та паралелізму даних. Він абстрагує базове апаратне забезпечення та дозволяє ефективно використовувати різнорідні ресурси.

SYCL (Single-source C++ for OpenCL) – це абстракція вищого рівня, створена на основі OpenCL. Вона дозволяє розробникам писати єдиний вихідний код на C++, що може базуватись на гетерогенних систем. SYCL використовує функції паралелізму C++ і бездоганно інтегрується з OpenCL. Він спрощує процес програмування, надаючи більш звичну модель програмування C++ і дає змогу розробникам виражати паралелізм за допомогою стандартних конструкцій C++.

CUDA (Compute Unified Device Architecture) – це паралельна обчислювальна платформа та модель програмування. Розроблена компанією NVIDIA спеціально для власних графічних процесорів. Вона забезпечує інтерфейс програмування для прискорених обчислень на GPU. CUDA розширює мову C/C++ спеціальними директивами для GPU, дозволяючи розробникам писати паралельний код. CUDA забезпечує детальний контроль над переміщенням даних і синхронізацією між центральним та графічним процесорами.

1.2 Виклики та переваги гетерогенних паралельних обчислень

Підхід гетерогенних паралельних обчислень пропонує низку переваг, зокрема покращену продуктивність, підвищену енергоефективність, прискорене виконання робочого навантаження та масштабованість. Однак, цей підхід також має певні проблемами, які необхідно вирішити, щоб повністю використати його потенціал. Глибоке розуміння особливостей гетерогенних обчислень матиме вирішальне значення для ефективного використання можливостей цієї технології у різних областях застосування.

Розробка програмного забезпечення для гетерогенних систем має кілька проблем, які розробники повинні подолати, щоб ефективно використовувати можливості різноманітних процесорів. До таких проблем можна віднести:

- Спеціальні моделі та мови програмування для різних типів процесорів.
- Балансування навантаження та розподіл завдань.
- Переміщення та синхронізація даних.
- Управління пам'яттю.
- Відлагодження та профілювання створених програм.

1.2.1 Управління пам'яттю та переміщенням даних

Гетерогенні системи складаються з різних процесорів, таких як центральні процесори, графічні процесори та FPGA, кожен із яких має власний виділений простір пам'яті, між якими потрібно налагоджувати комунікацію, розподіляти пам'ять та передавати інформацію. Цей процес потребує ефективного керуванніщенням, з метою максимізації продуктивність та мінімізації накладних витрати. Однією з головних проблем тут є різниця у просторах пам'яті. Оскільки дані часто потрібно передавати між різними процесорами, то це може спричинити затримку та обмежувати загальну пропускної здатності. Крім того, оптимальне розміщення даних у пам'яті має вирішальне значення для ефективного виконання. Різні процесори можуть мати різні шаблони доступу. Наприклад, у графічних процесорах може використовуватись спільний доступу до пам'яті.

Синхронізація та узгодженість між різними просторами пам'яті також також проблема, яку слід вирішувати. Забезпечення узгодженості даних у кількох процесорах може вимагати механізмів синхронізації та ретельної координації, щоб уникнути конкуренції та гарантувати коректність.

1.2.2 Балансування навантаження та оптимізація розподілу завдань

У гетерогенних системах наявність різноманітних процесорів створює потребу ефективного балансування навантаження та розподілу завдань. Тобто передбачається рівномірний розподіл задач обчислення між доступними процесорами для максимального паралелізму та мінімізації часу простою. Однак досягнення балансу навантаження в гетерогенній системі може бути складною задачею через варіації можливостей обробки, різні

ієрархії пам'яті та накладні витрати на зв'язок. Оптимізація розподілу передбачає врахування таких факторів, як вимоги до обчислень, залежності даних і доступні ресурси. Нерівномірний розподіл може призвести до недостатнього використання певних блоків обробки, спричиняючи вузькі місця продуктивності та збільшення часу виконання. Динамічне балансування навантаження часто необхідне для адаптації до мінливих умов системи. Моніторинг і аналіз продуктивності в режимі реального часу є необхідним для виявлення дисбалансу та прийняття обґрунтованих рішень щодо міграції або перерозподілу завдань.

1.2.3 Переваги гетерогенних систем

Гетерогенні системи пропонують ряд переваг перед традиційними гомогенними системами, забезпечуючи значне покращення продуктивності, енергоефективності та універсальності:

- Використання різних процесорів для підняття ефективності за рахунок паралельного виконання обчислень. Ефективний розподіл обчислювальних завдань між блоками виконання сприяє вищій пропускній здатності та швидшому часу виконання.
- Спеціалізовані блоки обробки в гетерогенних системах дозволяють підвищити загальну енергоефективності.
- Гетерогенну систему легко масштабувати. Ми можемо додати більше процесорів по мірі зростання обчислювальних вимог. Це особливо важливо для вирішення складних задач.

Розділ 2

Основи кількісного дизайну та аналізу

"Грам власного досвіду коштує дорожче тонни чужих повчань."

– Магатма Ґанді

Комп'ютерні технології значно розвинулися за останні 65 років завдяки технологічному прогресу та інноваціям у дизайні комп'ютерів. Мікропроцесори, стандартизовані операційні системи та RISC-архітектура відіграли значну роль у створенні простіших інструкцій, які підняли планку продуктивності. Ці досягнення призвели до щорічного зростання продуктивності на понад 50Мікропроцесори домінують у всьому спектрі комп'ютерного дизайну, від міні-комп'ютерів до високопродуктивних суперкомп'ютерів.

2.1 Огляд основних обчислювальних класів та їхніх системних характеристик

Зміни в комп'ютерних технологіях призвели до значних змін у тому, як комп'ютери використовуються та сприймаються в новому столітті. Це призвело до появи п'яти окремих комп'ютерних ринків, кожен з яких має унікальні сфери застосування, вимоги та технології.

Персональні мобільні пристрої (ПМП) – це бездротові пристрої з мультимедійними інтерфейсами, такі як мобільні телефони та планшетні комп'ютери. Вартість та енергоефективність є важливими факторами завдяки низькій споживчій ціні та використанню акумулятора. Додатки для ПМП часто базуються на мультимедійних даних і використовують флеш-пам'ять замість магнітних дисків. Для медіа-додатків з вимогами до продуктивності в реальному часі важлива оперативність і передбачуваність, а оптимізація пам'яті має вирішальне значення з огляду на її вартість. ПМП також вимагають енергоефективності через живлення від батареї та тепловіддачу.

Настільні комп'ютери. Ринок настільних комп'ютерів є найбільшим і охоплює як недорогі нетбуки, так і висококласні робочі станції. Більше половини настільних комп'ютерів, що виробляються щороку, - це ноутбуки, що працюють від батареї. Ринок прагне оптимізувати співвідношення ціни та якості, а поєднання продуктивності та ціни є важливим для споживачів і дизайнерів. Нові високопродуктивні мікропроцесори часто з'являються першими в настільних системах. Ринок настільних комп'ютерів добре охарактеризований з точки зору додатків та бенчмаркінгу. Однак, зростаюче використання веб-орієнтованих, інтерактивних додатків ставить нові виклики в оцінці продуктивності.

Сервери. У 1980-х роках сервери стали вирішальними для більш масштабних і надійних файлових та обчислювальних сервісів, замінивши мейнфрейми. Доступність є критично важливою для серверів, особливо для тих, що працюють з банкоматами або системами бронювання авіаквитків. Масштабованість є ще однією важливою характеристикою, оскільки серверні системи часто зростають у відповідь на збільшення попиту або

функціональних вимог. Нарешті, сервери призначені для ефективної пропускної здатності, яка вимірюється кількістю транзакцій за хвилину або веб-сторінок, що обслуговуються за секунду. Хоча швидкість реагування на індивідуальні запити є важливою, загальна ефективність та економічність є ключовими показниками для більшості серверів. Доступність, масштабованість і пропускна здатність є важливими характеристиками серверів.

Кластери. Поширення програмного забезпечення як послуги (SaaS) для таких додатків, як пошук, соціальні мережі, обмін відео та онлайн-покупки, призвело до розвитку кластерів і warehouse-scale computers (WSCs), які являють собою сукупність настільних комп'ютерів або серверів, об'єднаних локальними мережами. WSC дуже великі, з десятками тисяч серверів, які працюють як один комп'ютер. Співвідношення ціни, продуктивності та потужності є критично важливими для WSC, і вони покладаються на надлишкові, недорогі компоненти з програмним забезпеченням для виявлення та ізоляції збоїв. Доступність має вирішальне значення для WSC, які пов'язані з серверами, але масштабованість забезпечується локальною мережею, що з'єднує комп'ютери.

Вбудовані комп'ютери. Вбудовані комп'ютери, тобто мікропроцесори, можна знайти в повсякденній техніці, такій як мікрохвильові печі, пральні машини, принтери та автомобілі. Процесори в ПМД також вважаються вбудованими комп'ютерами, але їх виділяють в окрему категорію, оскільки вони можуть працювати з програмним забезпеченням, розробленим ззовні. Діапазон обчислювальних потужностей і вартості вбудованих комп'ютерів великий: деякі процесори коштують менше десяти центів, тоді як інші можуть виконувати мільярди інструкцій за секунду і коштують 100 доларів. Основною метою при розробці вбудованих процесорів часто є задоволення потреб у продуктивності за мінімальну ціну.

2.2 Класи паралелізму та паралельні архітектури

Паралелізм на декількох рівнях є рушійною силою комп'ютерного дизайну, причому основними обмеженнями є енергоспоживання та вартість. В додатках виділяють два типи паралелізму:

1. Паралелізм на рівні даних (DLP) виникає через те, що існує багато елементів даних, які можна оперувати одночасно.
2. Паралелізм на рівні завдань (TLP) виникає тому, що створюються робочі завдання, які можуть працювати незалежно і значною мірою паралельно.

Комп'ютерне обладнання, в свою чергу, може використовувати ці два види паралелізму додатків чотирма основними способами:

1. Паралелізм на рівні інструкцій використовує паралелізм на рівні даних на нижчих рівнях за допомогою компілятора, використовуючи такі ідеї, як конвеєризація, та на середніх рівнях, використовуючи такі ідеї, як конкурентне виконання.
2. Векторні архітектури та графічні процесори (GPU) використовують паралелізм на рівні даних, паралельно застосовуючи одну інструкцію до набору даних.
3. Паралелізм на рівні потоків використовує або паралелізм на рівні даних, або паралелізм на рівні задач у тісно пов'язаній апаратній моделі, яка дозволяє взаємодіяти між паралельними потоками.
4. Паралелізм на рівні запитів використовує паралелізм між розділеними завданнями, які визначені програмістом або операційною системою.

Ці чотири підходи апаратної підтримки паралелізму на рівні даних і паралелізму на рівні задач існують вже 50 років. Коли Майкл Флінн вивчав паралельні обчислення у 1960-х роках, він знайшов просту класифікацію, яку ми використовуємо і сьогодні. Він розглядав паралелізм на рівні інструкцій та даних, таким чином усі комп'ютери класифікують на чотири категорії:

1. Один потік інструкцій, один потік даних (SISD) - цей тип комп'ютера є однопроцесорним, який працює послідовно.

2. Один потік інструкцій, кілька потоків даних (SIMD) - цей тип комп'ютера виконує одну і ту ж інструкцію на кількох процесорах, використовуючи різні потоки даних. SIMD-машини використовують паралелізм на рівні даних для одночасного виконання одних і тих же операцій над кількома елементами даних. Кожен процесор має власну пам'ять даних, тоді як існує єдина пам'ять інструкцій і керуючий процесор, який отримує і відправляє інструкції.

3. Множинні потоки команд, один потік даних (MISD) - комерційних мультипроцесорів цього типу поки що не створено, але вони доповнюють цю класифікацію.

4. Кілька потоків інструкцій, кілька потоків даних (MIMD) - кожен процесор отримує свої власні інструкції і оперує своїми власними даними, орієнтуючись на паралелізм на рівні завдань. MIMD є більш гнучкою, ніж SIMD, і тому є більш універсальною, але за своєю суттю є більш дорогою. Комп'ютери MIMD також можуть використовувати паралелізм на рівні даних, але накладні витрати, ймовірно, будуть вищими, ніж у комп'ютерів SIMD. Ці накладні витрати означають, що розмір об'єму завдання повинен бути достатньо великим, щоб ефективно використовувати паралелізм.

2.3 Технічні тренди

Для того, щоб архітектура набору інструкцій процесора була успішною, вона повинна бути спроектована так, щоб витримати швидкі зміни в комп'ютерних технологіях. Зрештою, успішна архітектура набору інструкцій може існувати десятиліттями - наприклад, ядро мейнфрейма IBM використовується вже майже 50 років. Архітектор повинен планувати технологічні зміни які можуть збільшити термін служби комп'ютера.

Щоб планувати еволюцію комп'ютера, дизайнер повинен бути в курсі швидких зміни в технологіях реалізації. П'ять технологій реалізації компонентів є критично важливими:

- Логічна технологія інтегральних схем: Кількість транзисторів на мікросхемі подвоюється кожні 18-24 місяці завдяки щорічному зростанню кількості транзисторів на 40%-55%, що є результатом 35% щорічного збільшення щільності транзисторів та повільнішого і менш передбачуваного зростання (від 10% до 20% на рік) розміру кристала. Однак швидкість пристрою зростає не так швидко.

- Напівпровідникова DRAM (динамічна пам'ять з довільним доступом): Ємність DRAM на мікросхемі зростає приблизно на 25-40% щороку, подвоюючись приблизно кожні два-три роки. Однак темпи зростання сповільнюються, і є побоювання, що вони можуть зупинитися через проблеми з виробництвом ще менших комірок DRAM.

- Напівпровідникова флеш-пам'ять (електрично видаляюча програмована пам'ять тільки для читання): Ємність флеш-пам'яті на чіп подвоюється приблизно кожні два роки завдяки щорічному зростанню на 50-60%. Флеш-пам'ять набагато дешевша, ніж DRAM: у 2011 році вартість біта була в 15-20 разів нижчою.

- Технологія магнітних дисків: Щільність магнітних дисків подвоюється кожні три роки, починаючи з 2004 року, завдяки щорічному зростанню на 40%. Однак диски все ще набагато дешевші за флеш-пам'ять і DRAM, оскільки вартість біта в них у 300-500 разів нижча, ніж у DRAM. Магнітні диски широко використовуються в кластерних системах зберігання даних.

- Мережеві технології: Продуктивність мережі залежить від продуктивності комутаторів та системи передачі даних.

2.4 Потужність та енергоспоживання

Існують три важливі фактори, які слід враховувати при проектуванні або використанні процесора: максимальна потужність, тривале енергоспоживання та енергоефективність. Максимальна потужність, необхідна процесору, має вирішальне значення для забезпечення його належного функціонування, оскільки спроба споживати більше енергії, ніж може забезпечити система, може призвести до падіння напруги та збоїв у роботі.

Процесори передбачають методи індексації напруги для регулювання напруги в більш широких межах, але це знижує продуктивність. Стале споживання енергії, або теплова розрахункова потужність (TDP), важлива для визначення потреби в охолодженні, оскільки недостатнє охолодження може призвести до виходу пристрою з ладу і незворотних пошкоджень. Сучасні процесори мають функції управління тепловиділенням, включаючи зниження тактової частоти і вимкнення живлення чіпа, якщо це необхідно.

Енергоспоживання та енергоефективність також є важливими факторами при порівнянні процесорів. Енергоспоживання є кращим показником для порівняння процесорів, ніж потужність, оскільки воно прив'язане до конкретного завдання і часу, необхідного для його виконання. Порівняння енергоспоживання для виконання завдання є кращим способом визначення ефективності, ніж порівняння середньої потужності. Використання енергії для порівняння альтернатив особливо важливо, коли є фіксоване робоче навантаження, наприклад, для хмари або смартфона, оскільки споживання енергії визначає рахунок за електроенергію для хмари і термін служби акумулятора для смартфона. Потужність може бути корисним показником як обмеження, але справжньою метрикою для порівняння продуктивності є енергоспоживання.

2.5 Переваги паралелізму

Використання паралелізму є ключовим методом підвищення продуктивності. Один з прикладів - застосування паралелізму на системному рівні. Для підвищення пропускної здатності стандартного серверного тесту, такого як SPECWeb або TPC-C, можна використовувати кілька процесорів і дисків. Це дозволить розподілити навантаження по обробці запитів між процесорами і дисками, що призведе до збільшення пропускної здатності. Можливість розширення кількості процесорів, дисків і пам'яті називається масштабованістю і є цінною функцією для серверів. Розподіл даних на декількох дисках для паралельного читання і запису називається паралелізмом на рівні даних. SPECWeb використовує паралелізм на рівні запитів для використання декількох процесорів, тоді як TPC-C використовує паралелізм на рівні потоків для швидкої обробки запитів до бази даних.

На рівні одного процесора використання паралелізму між інструкціями має вирішальне значення для досягнення високої продуктивності. Конвеєризація є одним з найпростіших методів досягнення цієї мети. Фундаментальна концепція конвеєризації полягає у перекритті виконання інструкцій для зменшення загального часу, необхідного для завершення послідовності інструкцій.

Паралелізм також може бути використаний на рівні детального цифрового проектування. Наприклад, кеш-пам'ять використовує декілька розділів пам'яті, які зазвичай обшукуються паралельно, щоб знайти потрібний елемент. Сучасні АЛП (арифметико-логічні пристрої) використовують функцію перенесення з випередженням, яка використовує паралелізм для прискорення процесу обчислення сум від лінійних до логарифмічних за кількістю бітів на операнд.

Розділ 3

Масштабоване паралельне виконання

"Той, хто вчиться, але не мислить – втратить себе.
Той, хто мислить, але не вчиться – занапастить себе."

– Конфуцій (551 - 479 до н. е.)

Масштабоване паралельне виконання відноситься до одночасного виконання обчислень на багатьох ресурсах з метою покращення продуктивності та ефективності. Це передбачає розподіл завдань між обчислювальними пристроями, такими як процесори, ядра чи вузли кластерів, для прискорення обчислень, зменшення часу виконання та обробки великих обсягів даних. Масштабоване паралельне виконання вимагає розробки відповідного алгоритмічного та програмного забезпечення, а також управління ресурсами та ефективної комунікації між різними ресурсами.

Масштабованість є ключовою властивістю паралельних систем, оскільки вона визначає їх здатність ефективно працювати зі зростанням обчислювальних ресурсів. Масштабованість дозволяє системі збільшувати продуктивність пропорційно доданим ресурсам, що робить її більш гнучкою та потужною. Забезпечення масштабовості дозволяє розподіляти завдання між багатьма ресурсами, уникати перевантажень та забезпечувати більш швидке виконання обчислень. Відсутність масштабовості може призводити до погіршення продуктивності, витрат ресурсів та недосяжності поставлених цілей. Отже, розуміння та досягнення масштабовості є важливими аспектами при розробці та використанні паралельних систем.

Метою даного розділу є знайомство читача з фундаментальними особливостями масштабованого паралельного виконання. Ви зрозумієте концепцію масштабованості, її важливість в паралельних системах та роль, яку вона відіграє для ефективного виконання обчислень. У цьому розділі будуть розглянуті різні аспекти масштабованого паралельного виконання, включаючи архітектури, методи та стратегії, виклики та перешкоди, а також застосування в різних областях.

3.1 Основні поняття масштабованого паралельного виконання

3.1.1 Паралельність та масштабованість

Паралельність є ключовим аспектом масштабованого паралельного виконання. Вона відноситься до одночасного виконання багатьох завдань на різних обчислювальних ресурсах, таких як процесори, ядра чи вузли кластерів. Паралельність дозволяє розподілити завдання між ресурсами для покращення продуктивності та зменшення часу виконання. Масштабованість відноситься до здатності системи збільшувати продуктивність пропорційно зростанню ресурсів. Забезпечення паралельності та масштабованості є важливими аспектами розробки та використання паралельних систем для досягнення ефективного та швидкого виконання обчислень [1].

3.1.2 Закони Амдала та Густавсона

Закони Амдала та Густавсона є важливими концепціями, пов'язаними з масштабованим паралельним виконанням. Закон Амдала стверджує, що швидкість виконання програми на паралельній системі обмежується найповільнішою частиною програми, яка не може бути паралелізована. Закон Густавсона наголошує на використанні зростаючої кількості ресурсів для розв'язання більших проблем та підвищення продуктивності, навіть при фіксованій розмірності проблеми.

Обидва закони відіграють важливу роль у визначенні ефективності та можливостей масштабованого паралельного виконання. Розуміння цих законів дозволяє розробляти оптимальні стратегії паралельного програмування та планування ресурсів для досягнення більш ефективного виконання обчислень.

3.1.3 Види масштабованого паралельного виконання

Масштабоване паралельне виконання може бути реалізоване у різних формах залежно від способу організації та розподілу завдань. Декілька видів масштабованого паралельного виконання включають:

1. Загальнопаралельне виконання (MIMD): Цей підхід передбачає незалежне виконання різних завдань на окремих процесорах або вузлах кластера. Кожен процесор або вузол може виконувати свою власну послідовну програму.

2. Масштабоване виконання з використанням потоків (SIMD): У цьому випадку одна програма виконується паралельно на кожному процесорі, які виконують однакові операції, але над різними наборами даних.

3. Масштабоване виконання з використанням задач (Task Parallelism): Цей підхід передбачає розподіл великих завдань на більшу кількість менших, незалежних підзавдань, які можуть бути виконані паралельно на різних процесорах або вузлах.

4. Масштабоване виконання з використанням даних (Data Parallelism): У цьому випадку дані розподіляються між різними процесорами або вузлами, і кожен процесор або вузол обробляє свій власний піднабір даних паралельно з іншими.

Комбінація цих підходів може бути використана для досягнення більш ефективного та масштабованого паралельного виконання в залежності від конкретного завдання та архітектури системи.

3.2 Архітектури для масштабованого паралельного виконання

3.2.1 Симетрична багатопроцесорна архітектура (SMP)

Симетрична багатопроцесорна архітектура (SMP) є одним з популярних типів архітектур для масштабованого паралельного виконання. У цій архітектурі кілька процесорів мають спільний доступ до одного пам'яті та виконують програму паралельно. Кожен процесор може виконувати будь-яку частину програми, і дані можуть бути розподілені між процесорами для обробки.

SMP архітектура забезпечує високу гнучкість та простоту програмування, оскільки програми можуть бути розроблені без необхідності докладного розподілу завдань. Крім того, SMP системи легко масштабуються, оскільки нові процесори можуть бути додані до системи для збільшення її потужності.

Проте, SMP архітектура також має свої обмеження. Залежно від конкретної реалізації, вона може мати обмеження щодо кількості процесорів та шляхів доступу до спільної пам'яті, що може обмежити її масштабованість. Крім того, SMP системи можуть мати проблеми з конфліктами при одночасному доступі до спільних ресурсів, таких як кеш-пам'ять або шина.

Враховуючи переваги та обмеження, SMP архітектури застосовуються в різних областях, включаючи сервери, робочі станції та деякі вбудовані системи, де вимоги до паралельного виконання можуть бути задоволені шляхом використання кількох процесорів для підвищення продуктивності.

3.2.2 Розподілена пам'ять (Distributed Memory)

Розподілена пам'ять є одним з типів архітектур для масштабованого паралельного виконання. У такій архітектурі кожен процесор має власну приватну пам'ять, і комунікація між процесорами здійснюється за допомогою обміну повідомленнями. Кожен процесор працює над своєю власною часткою даних та виконує відповідну частину програми [1].

Розподілена пам'ять дозволяє масштабувати систему, додавати нові процесори та розподіляти завдання між ними. Кожен процесор може працювати незалежно, що дозволяє досягнути високої продуктивності у виконанні паралельних завдань. Проте, використання розподіленої пам'яті вимагає додаткових зусиль для керування комунікацією та синхронізацією між процесорами.

Такі архітектури часто використовуються в кластерах або суперкомп'ютерах, де великі обчислювальні завдання розподіляються між багатьма вузлами, і кожен вузол має власні процесори та пам'ять.

3.2.3 Гібридна архітектура (Hybrid)

Гібридна архітектура є комбінацією розподіленої пам'яті та симетричної багатопроцесорної архітектури. В цьому типі архітектури кілька вузлів, які мають свої власні процесори та пам'ять, підключені до спільної комунікаційної мережі.

Гібридна архітектура дозволяє комбінувати переваги розподіленої пам'яті та симетричної багатопроцесорної архітектури. Це дозволяє розподіляти завдання між вузлами та використовувати локальну пам'ять кожного вузла для зменшення навантаження на комунікаційну мережу. Крім того, гібридні архітектури можуть бути масштабовані, додаванням нових вузлів для збільшення потужності системи.

Цей тип архітектури використовується в різних областях, де потрібно обробляти великі обчислювальні завдання. Наприклад, вона широко застосовується в обчислювальних кластерах, суперкомп'ютерах та грід-системах, де вимоги до паралельного виконання можуть бути задоволені за допомогою комбінації різних типів архітектур.

3.3 Методи та стратегії масштабованого паралельного виконання

3.3.1 Розподілені обчислення

Розподілені обчислення - це методика, що використовується для розподілу обчислювальних завдань між різними вузлами або комп'ютерами в мережі. У такому підході, обчислювальні завдання розбиваються на менші підзадачі, які виконуються паралельно на різних вузлах, а результати об'єднуються для отримання кінцевого результату.

Розподілені обчислення дозволяють ефективно використовувати ресурси великих комп'ютерних систем, таких як кластери або грід-системи. Цей підхід забезпечує масштабованість, оскільки можна додавати нові вузли до мережі для збільшення потужності обчислень.

Такий метод застосовується в багатьох областях, включаючи наукові дослідження, обробку великих об'ємів даних, обчислення в реальному часі та інші сфери, де потрібні високопродуктивні обчислення. Розподілені обчислення дозволяють розподіляти завдання між різними вузлами, забезпечуючи швидкість та ефективність виконання паралельних обчислень.

3.3.2 Паралельне програмування

Паралельне програмування - це методика розробки програм, яка дозволяє виконувати обчислення паралельно на багатьох обчислювальних ресурсах, таких як процесори чи вузли кластера. Цей підхід використовується для досягнення високої продуктивності та швидкодії обчислень шляхом розподілу завдань між різними ресурсами.

Паралельне програмування включає в себе використання спеціальних програмних інтерфейсів (API) та бібліотек для керування паралельним виконанням, таких як OpenMP, MPI, CUDA, або OpenCL. Ці інструменти надають можливості для розподілу завдань, синхронізації та комунікації між паралельними обчислювальними одиницями.

Паралельне програмування є ключовим елементом для досягнення масштабованого паралельного виконання. Воно дозволяє розподіляти завдання між різними процесорами чи вузлами, виконувати їх паралельно та об'єднувати результати. Правильне використання паралельного програмування може покращити продуктивність та ефективність обчислень у паралельних системах.

3.3.3 Поділ задач та даних

Поділ задач та даних є ключовим етапом у масштабованому паралельному виконанні. Цей підхід передбачає розбиття обчислювальних завдань на менші підзадачі, які можуть бути виконані паралельно на різних обчислювальних ресурсах.

Поділ задач може бути здійснений по різних критеріях, наприклад, по функціональності або по часовій складності. Кожна підзадача незалежно виконується на окремому ресурсі, що дозволяє зменшити час виконання завдання.

Поділ даних означає розподіл вхідних даних між різними обчислювальними ресурсами. Це може бути розподіл по блоках даних, розподіл по рядках або розподіл по кускам даних. Кожен ресурс отримує свою частину даних для обробки.

Правильний поділ задач та даних є важливим для досягнення ефективного масштабованого паралельного виконання. Він дозволяє використовувати ресурси системи оптимально та забезпечує баланс навантаження між обчислювальними ресурсами.

3.3.4 Синхронізація та комунікація

Синхронізація та комунікація є важливими аспектами масштабованого паралельного виконання. Вони використовуються для забезпечення взаємодії та координації між обчислювальними ресурсами у паралельній системі.

Синхронізація включає в себе механізми, які забезпечують правильний порядок виконання паралельних задач та уникнення конфліктів даних. Це можуть бути блокування, семафори, м'ютекси або бар'єри, які забезпечують послідовність виконання задач та доступ до спільних ресурсів.

Комунікація передбачає обмін даними між різними обчислювальними ресурсами у паралельній системі. Це можуть бути передача повідомлень, обмін даними через спільну пам'ять або використання спеціальних комунікаційних протоколів. Комунікація необхідна для координації дій ресурсів та передачі результатів обчислень.

Ефективна синхронізація та комунікація грають важливу роль у досягненні високої продуктивності та швидкодії масштабованого паралельного виконання. Вони дозволяють ресурсам співпрацювати, обмінюватись необхідною інформацією та досягати глобальних цілей обчислень.

3.4 Виклики масштабованого паралельного виконання

3.4.1 Залежність задач та конфлікти

Залежність задач та конфлікти є викликами, з якими стикаються при масштабованому паралельному виконанні. Залежність задач виникає, коли одна задача залежить від результату іншої задачі, тобто потребує її виконання або даних, щоб продовжити свою роботу. Це може створювати бар'єри для паралельного виконання, оскільки задачі не можуть бути виконані одночасно, поки не виконаються їх залежності.

Конфлікти виникають, коли різні задачі або ресурси намагаються отримати доступ до спільних ресурсів або даних одночасно. Це може призводити до суперечок, неправильних результатів або блокування виконання задач. Вирішення конфліктів вимагає відповідних механізмів синхронізації та управління доступом до ресурсів.

Ефективне управління залежностями задач та розв'язання конфліктів є важливими аспектами масштабованого паралельного виконання. Це може включати використання методів розподілу завдань, редагування порядку виконання, асинхронного виконання або використання технологій транзакцій та блокування для управління конфліктами.

3.4.2 Скалірування та ефективність

Скалірування та ефективність є важливими аспектами масштабованого паралельного виконання. Скалірування означає здатність системи збільшувати пропорційно кількості ресурсів (процесорів, вузлів, ядер) та завдань для забезпечення покращення продуктивності. Ефективність вимірює, наскільки добре система використовує доступні ресурси для досягнення бажаних результатів.

Одним із викликів є досягнення лінійного або майже лінійного скалірування, де збільшення кількості ресурсів пропорційно покращує продуктивність. Це може бути ускладнено залежностями задач, комунікацією між ресурсами або іншими обмеженнями системи.

Ефективність вимагає оптимального використання ресурсів та мінімізації зайвого навантаження. Це може бути досягнуто за допомогою ефективних алгоритмів, оптимізацій пам'яті та комунікації, управління пам'яттю та процесами, а також використанням спеціалізованих бібліотек та інструментів.

Скалірування та ефективність є ключовими вимогами для успішного масштабованого паралельного виконання, оскільки вони визначають продуктивність та використання доступних ресурсів у системі.

3.4.3 Переносимість програм

Переносимість програм є важливим викликом при розробці масштабованих паралельних систем. Це означає здатність програми працювати на різних архітектурах, операційних системах та платформах без потреби великої кількості модифікацій.

Щоб забезпечити переносимість, програми повинні використовувати стандартизовані програмні інтерфейси, мови програмування або бібліотеки, що підтримуються різними системами. Крім того, важливо уникати залежностей від конкретних архітектурних особливостей або операційних систем.

Переносимість програм сприяє легкості розробки та управлінню системами, дозволяючи їх використання на різних комп'ютерах та платформах. Це особливо важливо в контексті масштабованого паралельного виконання, де системи можуть бути розгорнуті на різних обчислювальних ресурсах із різними характеристиками.

3.4.4 Управління ресурсами та балансування навантаження

Ефективне масштабоване паралельне виконання вимагає ефективного управління ресурсами та балансування навантаження між обчислювальними вузлами. Це включає в себе розподіл задач та даних між вузлами системи, а також оптимальне використання доступних ресурсів, таких як процесорний час, пам'ять та мережева пропускна здатність.

Управління ресурсами може включати в себе такі аспекти, як планування виконання задач, алгоритми розподілу ресурсів, контроль навантаження та управління пріоритетами задач. Балансування навантаження полягає в розподілі задач між вузлами системи таким чином, щоб уникнути перевантаження одних вузлів і недостатку ресурсів на інших.

Ефективне управління ресурсами та балансування навантаження дозволяє досягти оптимального використання ресурсів системи, підвищує продуктивність та швидкодію, а також забезпечує рівномірне розподілення

роботи між вузлами. Це сприяє покращенню масштабованості системи та забезпечує ефективне виконання паралельних обчислень.

3.5 Застосування масштабованого паралельного виконання

3.5.1 Високопродуктивні обчислення

Масштабоване паралельне виконання використовується в області високопродуктивних обчислень для розв'язання складних задач, які вимагають великої обчислювальної потужності. Це можуть бути наукові дослідження, обробка великих обсягів даних, симуляції складних процесів, моделювання, криптографічні обчислення та багато інших застосувань.

Високопродуктивні обчислення використовують паралельне виконання для розподілу задач між багатьма обчислювальними вузлами, що дозволяє зменшити час виконання завдань. Завдяки масштабованості паралельного виконання, можна використовувати велику кількість обчислювальних вузлів для розподілених обчислень та досягнення високої швидкодії та продуктивності.

Високопродуктивні обчислення забезпечують здатність ефективно вирішувати складні завдання, які вимагають великої кількості обчислень та обробки даних. Застосування масштабованого паралельного виконання дозволяє досягти високої продуктивності та розв'язувати проблеми, що раніше були недосяжні з точки зору обчислювальних можливостей.

3.5.2 Обробка великих об'ємів даних

Масштабоване паралельне виконання використовується для ефективної обробки великих обсягів даних. Завдяки паралельному виконанню, обчислення та обробка даних розподіляються між багатьма обчислювальними вузлами, що дозволяє прискорити час виконання і забезпечити швидкодію при роботі з великими об'ємами даних.

Обробка великих об'ємів даних може включати такі завдання, як аналіз даних, машинне навчання, глибинне навчання, обробка сигналів, геномічна аналітика та інші. Застосування масштабованого паралельного виконання дозволяє ефективно розподілити обчислення та оптимізувати роботу з великими обсягами даних, що дозволяє швидко здійснювати аналіз, витягувати корисну інформацію та зробити інформаційне прийняття на основі результатів обробки.

3.5.3 Моделювання та симуляція

Масштабоване паралельне виконання застосовується для моделювання та симуляції складних систем. Завдяки паралельному обчисленню, можна ефективно моделювати та аналізувати поведінку систем, таких як фізичні процеси, кліматичні зміни, фінансові ринки та інші.

Моделювання та симуляція використовуються для передбачення результатів, вивчення взаємодії компонентів системи, оптимізації рішень та виявлення недоліків. Застосування масштабованого паралельного виконання дозволяє розподілити обчислення між багатьма процесорами або вузлами, що забезпечує швидке та ефективне моделювання складних систем і здійснення симуляційних експериментів.

3.5.4 Машинне навчання та штучний інтелект

Масштабоване паралельне виконання є важливим інструментом для реалізації машинного навчання та розвитку штучного інтелекту. Великі об'єми даних та складні моделі вимагають потужних обчислювальних ресурсів для тренування та розгортання моделей. Застосування паралельного обчислення дозволяє розподілити завдання між багатьма процесорами чи вузлами, що прискорює процес навчання моделей та забезпечує високу швидкодіючість.

Масштабоване паралельне виконання також дозволяє ефективно використовувати глибокі нейронні мережі, які мають велику кількість параметрів та потребують інтенсивного обчислювального ресурсу. Завдяки паралельному виконанню, можна прискорити процес тренування та інференсу моделей, що дозволяє швидше реагувати на нові дані та виконувати складні завдання штучного інтелекту.

Масштабоване паралельне виконання відкриває широкі можливості для застосування машинного навчання та розвитку штучного інтелекту в різних галузях, включаючи медицину, фінанси, автономні системи, обробку природних мов та багато іншого.

3.6 Приклади та інструменти масштабованого паралельного виконання

3.6.1 MPI (Message Passing Interface)

MPI є стандартом для обміну повідомленнями між процесами у паралельних обчисленнях. Він широко використовується в масштабованих паралельних системах для розробки програм, які виконуються на багатьох вузлах. MPI дозволяє процесам спілкуватися та обмінюватися даними через повідомлення, що спрощує розподілене обчислення.

За допомогою MPI можна реалізувати різні форми паралельного виконання, включаючи розділення завдань, взаємодію процесів та синхронізацію їх дій. Він надає набір функцій та протоколів, які дозволяють програмістам створювати паралельні програми, які ефективно використовують розподілені обчислювальні ресурси.

MPI знаходить широке застосування в галузях, де потрібно виконувати великі розподілені обчислення, наприклад, у наукових дослідженнях, обробці даних, моделюванні та симуляції. Багато відкритих та комерційних програмних пакетів для паралельних обчислень базуються на MPI, що дозволяє розробникам використовувати його потужні можливості для виконання вимогливих обчислювальних завдань.

3.6.2 OpenMP (Open Multi-Processing)

OpenMP є стандартом для розробки багатопотокових програм. Він дозволяє розпаралелювати виконання програми шляхом розподілу роботи між потоками. OpenMP використовує директиви компілятора та бібліотеки для створення паралельних рішень.

За допомогою OpenMP можна легко розпаралелювати цикли, функції, обчислення на матрицях та інші частини програми. Він підтримує різні види розподілення роботи, такі як розподіл по кількості потоків, розподіл по блоках даних та інші. OpenMP дозволяє програмістам ефективно використовувати потужність багатопроцесорних систем та багатоядерних процесорів.

OpenMP є популярним інструментом у багатьох галузях, включаючи наукові дослідження, чисельні обчислення, обробку даних та моделювання. Він підтримується багатьма компіляторами та доступний для використання на різних платформах. OpenMP є простим у використанні та дозволяє програмістам зосередитися на розпаралелюванні коду, покращуючи продуктивність програми.

3.6.3 CUDA (Compute Unified Device Architecture)

CUDA є платформою та програмним інтерфейсом, розробленим компанією NVIDIA для паралельного виконання обчислень на їх графічних процесорах (GPU). Вона дозволяє програмістам використовувати потужність GPU для виконання широкого спектру завдань, включаючи наукові обчислення, графіку, машинне навчання та обробку великих об'ємів даних.

CUDA надає програмістам доступ до паралельних обчислень на GPU шляхом використання спеціального мови програмування, який розширює стандартні мови програмування, такі як C і C++. Вона надає можливості для створення паралельних обчислювальних ядер, розподіленого виконання завдань, обміну даними між CPU та GPU, а також оптимізації продуктивності програм.

CUDA є популярним інструментом у сферах наукових досліджень, графіки, машинного навчання та інших областях, де вимагається висока обчислювальна потужність. Вона дозволяє програмістам використовувати паралельні можливості GPU для значного прискорення обчислень, що робить її ефективним інструментом для виконання складних завдань.

3.6.4 Apache Hadoop та Apache Spark

Apache Hadoop і Apache Spark є популярними фреймворками для обробки та аналізу великих об'ємів даних в розподілених середовищах.

Apache Hadoop надає інфраструктуру для збереження та обробки великих об'ємів даних на кластері з високою доступністю. Він базується на моделі розподіленої файлової системи Hadoop Distributed File System (HDFS) і фреймворку для обробки даних MapReduce. Hadoop дозволяє розподіляти обчислювальні завдання по вузлах кластера, забезпечуючи масштабованість та відмовостійкість.

Apache Spark є фреймворком для обробки даних в реальному часі та паралельного виконання розподілених обчислень. Він працює поверх Hadoop та надає більш високорівневий інтерфейс для програмування, що дозволяє розробникам ефективно використовувати розподілені можливості кластера. Spark підтримує широкий спектр операцій з даними, включаючи машинне навчання, обробку потокових даних та SQL-запити.

Apache Hadoop і Apache Spark є потужними інструментами для обробки великих об'ємів даних у розподілених середовищах. Вони дозволяють виконувати паралельні обчислення на великому масштабі та забезпечують високу продуктивність та масштабованість для різноманітних завдань обробки даних.

Розділ 4

Паралелізм на рівні команд (ILP): визначення та задачі

"Small daily improvements over time create stunning results."
"Маленькі щоденні покращення – ключ до приголомшливих довгострокових результатів."

– Робін Шарма

4.1 Поняття та представлення ILP на рівні архітектури комп'ютера

Процесори, зпроєктовані починаючи приблизно з 1985 року, можуть використовувати технологію пайплайнінгу ("конвеєра") для одночасного виконання декількох інструкцій і підвищення продуктивності та швидкості обчислень [2]. Така модель виконання команд називається паралелізмом на рівні інструкцій (команд) (ILP), оскільки команди у даній моделі можна виконувати паралельно. Вирізняють два підходи до імплеентації ILP, які значною мірою відрізняються один від одного:

1. Підхід, який покладається на апаратне забезпечення для динамічного виявлення та використання паралелізму.
2. Підхід, який покладається на технологію програмного забезпечення для статичного пошуку паралелізму під час компіляції.

Процесори, що використовують динамічний апаратний підхід, включаючи всі останні процесори Intel і багато процесорів ARM, переважають на ринках настільних ПК і серверів. На ринку персональних мобільних пристроїв ті самі підходи використовуються в процесорах, які є в планшетах і мобільних телефонах високого класу. У сфері Інтернет Речей, де обмеження потужності та вартості переважають над необхідністю продуктивності, розробники використовують нижчі рівні паралелізму на рівні інструкцій.

Одним з ключових понять ILP є поняття CPI (цикл на інструкцію). Для конвеєрного процесора CPI є сумою базового (ідеального) CPI та всіх циклів що виникають від затримок. Іншим поняттям ключовим для розуміння концепцій ILP є поняття базового блоку — прямолінійній послідовності без розгалужень на вході, окрім самого входу, і без розгалужень на виході, окрім виходу. Для типових програм RISC процесору середня динамічна частота розгалужень становить від 15% до 25%, що означає, що між парою розгалужень виконується від трьох до шести інструкцій.

4.2 Розгортання циклу(loop unrolling)

Проста схема для збільшення кількості інструкцій відносно розгалужень і накладних інструкцій - це розгортання циклу. Розгортання просто повторює тіло циклу кілька разів, регулюючи код завершення циклу. Розгортання циклу також можна використовувати для покращення планування. Оскільки він усуває розгалуження, він дозволяє планувати разом інструкції з різних ітерацій. У цьому випадку ми можемо усунути зупинки використання даних, створивши додаткові незалежні інструкції в тілі циклу. Якби ми просто повторили інструкції під час розгортання циклу, використання тих самих регістрів могло б перешкодити нам ефективно планувати цикл. Таким чином, ми можемо використовувати різні регістри для кожної ітерації, збільшуючи необхідну кількість регістрів.

4.2.1 Предиктор

У комп'ютерній архітектурі предиктор (передбачувач) розгалужень — це цифрова схема, яка намагається вгадати, яким шляхом піде розгалуження (наприклад, структура if–then–else) до того, як це буде відомо остаточно. Метою предиктора розгалужень є покращення потоку в конвеєрі інструкцій. Прогнози розгалужень відіграють вирішальну роль у досягненні високої продуктивності в багатьох сучасних конвеєрних архітектурах мікропроцесорів.

Двостороннє розгалуження зазвичай реалізується за допомогою інструкції умовного переходу. Умовний стрибок можна або «здійснити» та перейти в інше місце в пам'яті програми, або його можна «не зробити» та продовжити виконання одразу після умовного переходу. Невідомо напевно, чи буде виконано умовний стрибок чи ні, доки умова не буде обчислена і умовний стрибок не пройде стадію виконання в конвеєрі інструкцій.

Без передбачення розгалуження процесору довелося б чекати, доки інструкція умовного переходу не пройде стадію виконання, перш ніж наступна інструкція зможе перейти на стадію вибірки в конвеєрі. Прогноз розгалуження намагається уникнути цієї втрати часу, намагаючись вгадати, чи умовний стрибок найімовірніше буде виконано чи ні. Гілка, яка вважається найбільш ймовірною, потім вибирається та спекулятивно виконується. Якщо пізніше буде виявлено, що припущення було неправильним, тоді спекулятивно виконані або частково виконані інструкції відкидаються, і конвеєр починається заново з правильною гілкою, викликаючи затримку.

Час, втрачений у разі помилкового передбачення розгалуження, дорівнює кількості етапів у конвеєрі від етапу отримання до етапу виконання. Сучасні мікропроцесори, як правило, мають досить довгі конвеєри, тому затримка помилкового прогнозування становить від 10 до 20 тактів. Як наслідок, збільшення довжини конвеєра збільшує потребу в більш просунутому предикторі розгалужень.

Коли вперше зустрічається інструкція умовного переходу, немає багато інформації, на основі якої можна було б зробити прогноз. Але предиктор гілок веде записи про те, чи гілки зайняті чи ні. Коли він стикається з умовним стрибком, який бачили кілька разів раніше, тоді він може базувати прогноз на історії. Прогноз розгалуження може, наприклад, розпізнати, що умовний стрибок виконується частіше, ніж ні, або що він виконується кожного другого разу.

Прогноз розгалуження – це не те саме, що передбачення цільового розгалуження. Прогноз розгалуження намагається вгадати, чи буде здійснено умовний стрибок чи ні. Передбачення цілі розгалуження намагається вгадати ціль здійсненого умовного або безумовного стрибка до того, як вона буде обчислена шляхом декодування та виконання самої інструкції. Прогнозування розгалуження та цільове передбачення розгалуження часто поєднуються в одній схемі [3].

4.2.2 Інші методи

Виконання з попереднім аналізом (Speculative Execution). Цей метод використовується для забезпечення неперервного виконання інструкцій, навіть якщо не всі залежності між ними відомі заздалегідь. Процесор може спробувати виконати інструкцію з попереднім аналізом та робити припущення про значення регістрів або пам'яті, щоб продовжити виконання. Якщо аналіз був неправильним, виконання скасовується, але якщо він

був правильним, це дозволяє ефективно використовувати паралельне виконання інструкцій. Виконання з довільним порядком (Out-of-Order Execution) Цей метод дозволяє виконувати інструкції в будь-якому порядку, не обов'язково в послідовному порядку їх появи у програмі. Процесор може аналізувати залежності між інструкціями та виконувати ті, що не мають залежностей, в першу чергу. Це дозволяє заповнити "порожні" цикли процесора та забезпечує більш ефективне використання ресурсів. Ці приклади демонструють різні способи реалізації ILP у комп'ютерах. Комбінація цих методів та інших технік дозволяє досягти високої продуктивності та ефективного використання ресурсів процесора. Отже, Усі попередні розділи розкривають значення та основні концепції Instruction-Level Parallelism (ILP) в сучасних комп'ютерах. ILP використовується для досягнення паралельного виконання інструкцій та покращення продуктивності системи. Це досягається шляхом розробки та використання різних методів, таких як суперскалярна архітектура, векторні процесори, виконання з попереднім аналізом та виконання з довільним порядком. Реалізація ILP дозволяє ефективно розподіляти інструкції між різними функціональними блоками процесора, використовувати паралельне виконання та перекривати дії для досягнення більшої швидкості та ефективності. Використання ILP також допомагає ефективно використовувати ресурси процесора, забезпечувати швидку обробку завдань різного характеру та досягати високої продуктивності в сучасних комп'ютерах. Загалом, реалізація ILP відіграє важливу роль у покращенні продуктивності та ефективності обчислювальних систем, дозволяючи виконувати інструкції паралельно та ефективно використовувати ресурси процесора. Це сприяє зростанню швидкості обробки даних, поліпшенню виконання програм та розширенню можливостей сучасних комп'ютерів.

Література

[1] D. B. Kirk and W. H. Wen-Mei, *Programming massively parallel processors: a hands-on approach.* Morgan kaufmann, 2016.

[2] J. L. Hennessy and D. A. Patterson, *Computer architecture: a quantitative approach.* Elsevier, 2011.

[3] Wikipedia. Branchpredictor. Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Branch_predictor