МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
КОНОТОПСЬКИЙ ІНСТИТУТ

Кафедра електронних
приладів і автоматики

Кваліфікаційна робота бакалавра

АНАЛІЗ ТЕХНОЛОГІЇ БАГАТОЯДЕРНОГО ЦЕНТРАЛЬНОГО ПРОЦЕССОРА

Студент групи ЕІ – 61 І.Ковальов

Науковий керівник Ю.В.Столярчук

Нормоконтроль,
ст. викладач, к.т.н. О.Д. Динник

Конотоп 2020

РЕФЕРАТ

Об’єктом дослідження кваліфікаційної роботи є аналіз технології багатоядерного центрального процесора.

Мета роботи полягає у аналізі технології що використовуються при побудові та створенню багатоядерного центрального процесора.

При виконанні роботи було проаналізовано види, модифікації, переваги та недоліки технології багатоядерного центрального процесора, визначені фірми які є передовими в розробці багатоядерних процесорів, їхні особливості та принципи роботи з системою.

У результаті проведення аналізу встановлено, що багатоядерний процесор є важливим доповненням у часовій шкалі мікропроцесорів та засобом вирішення проблем з високим споживанням енергії, ціною та розміром. Виділено архітектуру Intel та AMD, двох найважливіших постачальників сучасних багатоядерних систем, і порівняння зроблено з точки зору архітектури

Робота викладена на _30_ сторінках, у тому числі включає _9_ рисунків, список цитованої літератури із _19_ джерел.

КЛЮЧОВІ СЛОВА: ЯДРО, ПРОЦЕСОР, БАГАТОЯДІРНІСТЬ, ШВИДКІСТЬ РОБОТИ, ОПЕРАЦІЙНА СИСТЕМА

ЗМІСТ

ВСТУП

За останні кілька десятиліть продуктивність універсальних процесорів зросла експоненціально, і вдосконалення дизайну та архітектури було результатом великого рівня. Оскільки обчислювальна техніка стає всюдисущою, сьогодні розробляються все складніші додатки. Через появу таких додатків, необхідність більш швидкої системи є важливою.

Для вирішення питань щодо продуктивності нещодавно введено поняття багатоядерності, де декілька ядер обробки оброблені до одного штампу. [1]

Багатоядерні процесори представляють багато переваг продуктивності, таких як кеш-пам'ять ядра, зменшене енергоспоживання та менший розмір. Однак збільшення кількості ядер на одному чіпі також представляє деякі проблеми, такі як пропускна здатність спільної пам'яті, узгодженість кеш-пам'яті, вміст кешу та проблеми з плануванням багатоядерних процесорів.

Наскільки відомо, багатоядерна архітектура недостатньо добре відображена в літературі, і досі не було зроблено жодного архітектурного порівняння. У цій роботі розглянуто архітектурні проекти та порівняння процесорів Intel та AMD.

Також висвітлюються потенційні проблеми багатоядерних систем, драйверів для багатоядерних систем, проблеми, які постають перед спільнотою програмного забезпечення появою багатоядерних технологій, різні варіанти програмного забезпечення та те, як програмне співтовариство, можливо, реагує на проблеми.

РОЗДІЛ 1
БАГАТОЯДЕРНІ ПРОЦЕСОРИ

1.1. Перед умови появі багатоядерних процесорів

Останнім часом у законі Мура спостерігається помітне уповільнення. Закон Мура говорить, що кількість транзисторів буде подвоюватися за кожні 18 місяців. Але, кількість транзисторів подвоюється, а продуктивність не йде. Продуктивність зберігалася до 2002 року завдяки таким технологіям:

- конвеєрне керування;
- кешування;
- надскалярний дизайн.

Однак після цього розрив став помітним, оскільки віддача від цих технологій почала давати зменшувані прибутки. На рис.1.1 зображено графік згідно закону Мура.

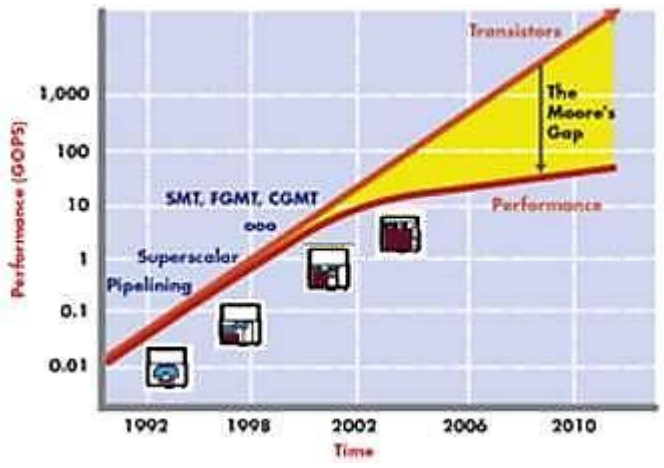

Рис.1.1. Розрив між продуктивністю та кількістю транзисторів [2]

Наприклад, між 1993 та 1999 роками швидкість процесора зросла в 10 разів. Перший процесор 1 ГГц був випущений у 2000 році, але за останні 9 років він збільшився лише до 3,3 ГГц, зростання значно повільніше, ніж за попередні шість років.

Споживана потужність пов'язана з частотою, а збільшення частоти робить величезні витрати на потужність. Проблеми проектування відповідних теплових синхронізацій, потоків повітря на серверах та настільних комп’ютерах стають вираженими зі збільшенням частоти. Іншим впливом є високі операційні витрати центрів обробки даних через більш високі витрати на системи кондиціонування та охолодження, а також існує тиск для зменшення витрат на експлуатацію підприємств та постачальників послуг. Це називається "стіна живлення". [3]

Використовуючи безліч методів оптимізації та вдаривши енергетичну стіну, напівпровідникові компанії зараз пакують більше ядер у процесор, замість того, щоб збільшувати швидкість процесорів. Перші версії багатоядерних процесорів підтримували технологію під назвою Hyper Threading.

Це дозволило два або більше потоків контексту, але вони поділили один кеш і зовнішню шину. Першим справжнім багатоядерним процесором став двоядерний процесор Intel Pentium, випущений у 2005 році.

Останні вироби від Intel quad core. Нові ядра ATOM та Cortex A9 підтримують декілька ядер та орієнтовані на ринки смартфонів та нетбуків. Лідером у багатоядерному є Tilera, який має процесори, що підтримують до 64 ядер. Існують прогнози, що найближчим часом процесори підтримуватимуть 100 ядер.

1.2. Паралельне програмування

Наслідок підвищення продуктивності перейшов від апаратного до програмного забезпечення, і головне питання полягає в тому, чи готовий до цього програмний світ. Якщо програмне забезпечення повинно ефективно використовувати декілька ядер, воно має перейти до моделі паралельного програмування, де кожне ядро працює незалежно від інші ядра. [4]

Парадигми паралельного програмування не є новинкою у світі програмного забезпечення. Існують класичні підручники, написані двадцять років тому про те,

як вирішити паралельне програмування. Існують такі «бентежно паралельні» операції:

- матричне множення;

- швидкі перетворення Фур'є;

- графічне відображення, які ідеально підходять для паралельного програмування,

- моделі програмування, такі як SIMD.

Але тривалий час паралельне програмування залишалося в межах мейнстріму, залишаючись цікавою для науковців, які працюють у таких областях, як прогнозування погоди, моделювання та розробник зброї в національних лабораторіях.

Поява багатоядерних процесорів зробила світ іншим. Підвищення швидкості в процесорах закінчився, і підвищення продуктивності повинно виходити з програмного забезпечення. На жаль, новини не дуже зручні з боку програмного забезпечення, і громада на жаль не готова до переходу.

З точки зору програмного забезпечення, є три аспекти:

- розробити нове паралельне програмне забезпечення;

- перефабрикувати існуючий код для отримання продуктивності;

- уникнути процесорного / інструментального / платформового замок в процесі. [5]

Проблема виражена для існуючого програмного забезпечення, яке вже використовується.

1.3 Функціональне програмування

Коли два або більше потоків отримують доступ до даних, завжди є ймовірність неправильного послідовності доступу та створення умови для перегонів. Щоб запобігти умовам перегонів, очікується, що програмісти

заблокують доступ до спільного ресурсу блокуванням. Але паралельне програмування схильне до помилок і може спричинити колізію в системі.

Важко уявити взаємодію потоків виконання після того, як кількість потоків виходить за межі декількох чисел. Складність порядку доступу та послідовностей зростає експоненціально, і людський розум не звик паралельно мислити.

Щоразу, коли код змінюється, завжди є можливість введення помилок. З введенням декількох потоків і одночасності, існує майже визначеність введення помилок. Коли програма виконується послідовно, певні припущення приймаються як належні, головне з них полягає в тому, що код буде виконаний у порядку, представленому в програмі. Після розбиття програми на процесори це припущення вже не дійсне і впевненість у коді nosedives.

У інженерному співтоваристві обговорюються дві пропозиції: повністю перейти до паралельного програмування за допомогою «Функціонального програмування» або використовувати інструменти з існуючими методами програмування для вирішення проблеми. Як ми бачимо, головна проблема паралельного програмування - паралельний доступ до поділилися даними та ускладненнями, що виникають із них. Усі процедурні мови (наприклад, C / C ++ / Java) страждають від однієї проблеми.

Існує думка, яка виступає за те, що повинні перейти до парадигми «функціонального програмування», щоб уникнути самої проблеми. Мови функціонального програмування мають своє коріння у галузі теоретичної інформатики під назвою "Lambda Calculus", винайденій Аланом Церквою у 1940-х роках. Це перетворилося на мову під назвою LISP, а функціональні мови характеризуються концепцією трактування всіх обчислень як оцінювання функції, а функціональні мови уникають стану та змінних даних. [6]

Не існує поняття глобальних змінних і функції не мають побічних ефектів. Усі відповідні дані зберігаються в стеку і є локальними для функції. З цієї причини функціональне програмування ідеально підходить для розподілених обчислень на

декілька ядер. І програмування на таких мовах гарантує, що не потрібно турбуватися щодо перегонових умов, тупикових ситуацій тощо.

Усі програмісти звикли до процедурного способу програмування та вивчення LISP або інших функціональних мов непросто, оскільки він має круту криву навчання. Як і об'єктно-орієнтоване програмування, потребує зміни «мислення», потрібно змінити спосіб мислення.

Для вирішення цього питання запропоновано деякі "підходи на середині". Сюди входить Scala ”, який є гібридом Java / FP8. Scala дозволяє програмісту використовувати існуючий код Java, використовувати знайомі плагіни Eclipse та компілювати код Scala до байтового коду Java. Аналогічно, Microsoft розробила середовище F # для .NET. Ці гібридні моделі дозволяють випробувати деякі модулі для функціонального програмування, не залишаючи звичного середовища існуючих бібліотек та коду.

Для користувачів функціональним програмуванням є такі варіанти, як Ерланг. Це мова, яку широко використовує Еріксон для своїх критично важливих продуктів. Але тривала крива навчання та академічний характер цих мов ускладнюють їх широке засвоєння.[7]

Замість того, щоб створювати революції і сподіватися, деякі компанії створюють інструменти, які можуть допомогти рефакторировать існуючий код для отримання кращої продуктивності. Ці інструменти розглядають існуючий код та пропонують оптимізацію та допомагають програмісту переробити кодекс для отримання продуктивності.

РОЗДІЛ 2
ІНСТРУМЕНТИ ТА ПРОГРАМНІ СЕРЕДОВИЩА ЯКІ ДОЗВОЛЯЮТЬ ПРАЦЮВАТИ З БАГАТОЯДЕРНИМИ ПРОЦЕСОРАМИ

2.1 Програмний інструмент RapidMind

RapidMind - це інструмент, який допомагає паралелізувати існуючий код для використання декількох ядер. Код потрібно змінити за допомогою конструкцій, які перекомпілюються інструментом. RapidMind який зображено на рис.2.1 було придбано Intel в серпні 2009 року.

До плюсів цього інструмента входить можливість орієнтації на різні архітектури (наприклад, nVidia, x86, процесори стільникового зв'язку) та можливість користуватися натурними ланцюжками інструментів (Windows VC ++ або Linux GNU ланцюги інструментів). Немає блокування в архітектурі процесора, оскільки код отримує повторне запрошення за різні періоди виконання.

У продукті стверджується, що він демонструє вражаючі результати в роботі, а пакет постачається з інструментами аналізу продуктивності та налагодження. Мінуси включають підтримку лише мови C ++ та блокування інструменту.

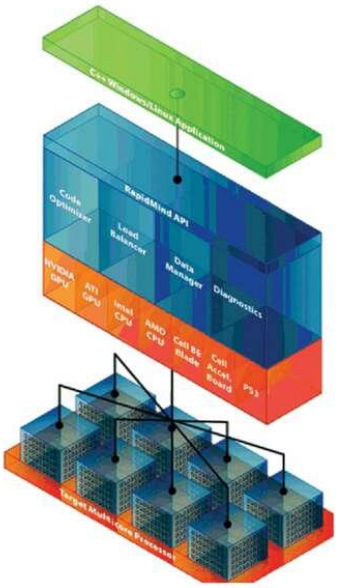

Рис.2.1. Архітектура інструменту RapidMind [8]

2.2 Програмний інструмент OpenMP

OpenMP - стандарт для багатопрограмування спільної пам'яті і визначений для мов C / C ++ / Fortran. Він складається з директив компілятора, бібліотеки виконання та змінних середовища. Код оснащений директивами, і він компілюється за допомогою компілятора, що підтримує openMP. Стандарт OpenMP зображений на рис.2.2.

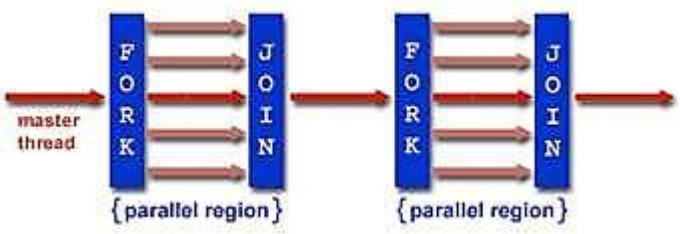

Рис.2.2. Стандарт OpenMP в роботі [9]

Стандарт дозволяє автоматично створювати декілька ниток, розгинаючи та з'єднуючись автоматично. Це дуже корисно в таких операціях, як маніпуляція з масивом. Приклад коду зображений на рис 2.3.

```
int main(int argc, char **argv) {
    const int N = 100000;
    int i, a[N];

    #pragma omp parallel for    <- Compiler directive
    for (i = 0; i < N; i++)
        a[i] = 2 * i;

    return 0;
}
```

Рис.2.3. Приклад коду що приписується при стандарті OpenMP [10]

До плюсів OpenMP належить:

- відносна легкість прийомів;
- приховування семантики потоку;
- можливість поступової паралелізації в коді;
- підтримка широкого спектру платформ і мов;

-підтримка грубого / тонкозернистого паралелізму.

Мінуси включають потребу в ланцюгах інструментів (компілятори, час виконання), які підтримують openMP. Ланцюг інструментів Sun Studio та GNU 4.3.1 підтримують openMP.OpenMP популярний серед математичних / наукових спільнот.

2.3 Програмне середовище розробки CUDA

CUDA - середовище розробки, яку сприяє nVidia13. Ця модель дозволяє масштабувати паралельну модель програмування на тисячах ниток. Як і OpenMP, модель пропонує розширення на мови C / C ++. Набір потоків групується як "блок потоків", і вони працюють разом, отримуючи доступ до частини спільної пам'яті, а кожен набір потоків відображається в блок потоку. Існують розширення для синхронізації та блокування, і CUDA призначений для процесорів nVidia і передбачає велику кількість кривої навчання. [10]

CUDA приймається в галузі наукових обчислень, візуалізації, графіки високого класу, фінансового моделювання та цифрової обробки сигналів. Поза науковими обчисленнями, в даний час інтерес до цієї моделі мало, оскільки вона щільно пов'язана з сімейством процесорів.

Ct - відповідь Intel на CUDA nVidia. Ct підтримує розширення мови C ++ як частина бібліотеки STL. Ця модель повинна забезпечити детермінізм, щоб уникнути тупиків і перегонів. Ланцюжок інструментів (очевидно) оптимізований для процесорів Intel x86. Це продукт, який все ще знаходиться в стадії розробки. Різні сегменти ринку програмного забезпечення, такі як сервер, наукові обчислення та настільні обчислення, швидше за все, застосовують різні стратегії для багатоядерних процесорів а саме:

- настільні програми, такі як електронні таблиці, текстові процесори чи редактори, не піддаються багатократній нарізці. Додано величезних зусиль, щоб переробити код і змусити їх використовувати потужність паралельної обробки.

Цілком ймовірно, що основне робоче програмне забезпечення не буде відремонтовано в значній мірі, і вони продовжуватимуть працювати так, як вони працюють сьогодні;

- існують деякі «бентежно паралельні» програми, які можна переписати за допомогою таких методів, як OpenMP / RapidMind / CUDA, і вони ввійдуть в мейнстрім. Одним із прикладів є візуалізація високого класу, де кожен піксель може бути відображений паралельно, і він надає можливість паралельного програмування. Аналогічно, при обробці пакетів кожне ядро може обробляти вхідний пакет незалежно. Такі програми легко масштабуються на декількох ядрах. Інший приклад - «перекодування», коли відео, закодоване в одному форматі, потрібно змінити на інший формат у режимі реального часу для потокового передавання;

- нові мови, такі як Ерланг, або гібриди типу Scalia / F # будуть перевірені в конкретних умовах, але широкомасштабне програмування та розгортання навряд чи відбудеться в найближчому майбутньому, враховуючи необхідний зміна парадигми;

- з боку робочого столу нові операційні системи, такі як Windows 7, дозволяють забезпечити функції афінності процесора. Вони дозволяють користувачеві приєднати додаток до ядра, наприклад, антивірусне програмне забезпечення може працювати на одному ядрі, відтворення мультимедіа на одному ядрі та решта програм на інших ядрах. Ця паралелізація грубого зерна покращить досвід роботи користувачів на робочих столах;

- збоку сервера технологія віртуалізації дозволить кожному ядру запускати різні зображення або операційні системи. Один може мати ядро під керуванням Windows Server, інший із запущеним Linux 2.6 та третій із запущеним застарілим додатком на Windows 95. Буде хороше використання багатоядерних систем на стороні сервера, але це буде на більш грубому рівні самої ОС, а не ніж на рівні додатків;

- у вбудованому світі можна перефактурувати код. Можливо також логічно розділити код на великі площі та працювати на різних ядрах. Наприклад, у когось може бути гіпервізор, вбудований код запускається на одному ядрі, а Windows працює на іншому ядрі. Дані, надані вбудованим кодом, можуть бути візуалізовані Windows, і вони можуть знизити загальну вартість системи.

2.4 Технологія Hyper-Threading

Технологія Intel Hyper-threading дозволяє кожному ядру процесора виконувати два завдання одночасно, по суті, роблячи з одного реального ядра два віртуальних. Зі слів автора [11]: “це можливо через те, що в таких ядрах зберігається стан відразу двох потоків, так як у ядра є свій набір регістрів, свій лічильник команд і свій блок роботи з перериваннями для кожного потоку. В результаті, операційна система бачить таке ядро, як два окремих ядра, і буде з ними працювати так само, як працювала б з двоядерним процесором. Однак інші елементи ядра для обох потоків - загальні, і діляться між ними. Крім цього, коли з якої-небудь причини один з потоків звільняє елементи конвеєра, інший потік використовує вільні блоки” .

Елементи конвеєра можуть бути не задіяні, якщо, наприклад, стався промах при зверненні в кеш-пам'ять, і необхідно вважати дані з ОЗУ, або невірно був передбачений перехід, або очікуються результати обробки поточної інструкції, або якісь блоки взагалі не використовуються при обробці даної інструкції.

Згідно джерела [11] : “більшість програм не можуть повністю навантажити процесор, так як деякі, в основному, використовують нескладні цілочисельні обчислення, практично не задіюючи блок FPU. Інші ж програми, наприклад 3D-студія, вимагають масу розрахунків з використанням чисел з плаваючою точкою, але при цьому звільняючи деякі інші виконавчі блоки і так далі”.

До того ж практично у всіх програмах - багато умовних переходів і залежних змінних. В результаті, використання технології Hyper-threading може дати істотний приріст продуктивності, сприяючи максимальному завантаженні конвеєра ядра.

Але не все так просто. Природно, приріст продуктивності буде менше, ніж від використання декількох фізичних ядер, так як все-таки потоки використовують загальні блоки одного конвеєра і часто змушені чекати звільнення необхідного блоку. До того ж більшість процесорів вже мають кілька фізичних ядер, і при використанні технології Hyper-threading віртуальних ядер може стати занадто багато, особливо, якщо процесор містить чотири і більше фізичних ядра.

Так як на даний момент програм, здатних розподіляти обчислення на велику кількість ядер, - вкрай мало, то в цьому випадку результат може розчарувати користувачів.

Є ще одна серйозна проблема технології Hyper-Threading - це конфлікти, що виникають, коли інструкції різних потоків потребують однотипних блоках. Може скластися ситуація, коли паралельно працюватимуть два схожих потоку, часто використовують одні й ті ж блоки. В такому випадку приріст продуктивності буде мінімальний.

В результаті, технологія Hyper-Threading дуже залежна від типу навантаження на процесор і може дати хороший приріст продуктивності, а може бути практично даремною.

2.5. Технологія Turbo Boost

Продуктивність більшості сучасних процесорів в домашніх умовах можна трохи підняти, просто кажучи розігнати - змусити працювати на частотах, що перевищують номінальну, тобто заявлену виробником.

Частота процесора розраховується, як частота системної шини, помножена на якийсь коефіцієнт, званий множником. Наприклад, процесор Core i7-970 працює з

системною шиною DMI на базовій частоті - 133 МГц, і має множник - 24. Таким чином, тактова частота ядра процесора складе: 133 Мгц * 24 = 3192 Мгц.

Якщо в налаштуваннях BIOS збільшити множник або підняти тактову частоту системної шини, то тактова частота процесора збільшиться, а, відповідно, збільшиться і його продуктивність. Однак процес цей - далеко небезпечний. Через розгону процесор може працювати нестабільно або взагалі вийти з ладу. Тому до розгону потрібно підходити відповідально і ретельно контролювати параметри роботи процесора.

З поява технології Turbo Boost все стало набагато простіше. Процесори з цією технологією можуть самі динамічно, на короткий проміжок часу, підвищувати тактову частоту, тим самим, збільшуючи свою продуктивність. При цьому процесор контролює всі параметри своєї роботи: напруга, силу струму, температуру і т.д., не допускаючи збоїв і тим більше виходу з ладу. Наприклад, процесор може відключити невживані ядра, тим самим, знизивши загальну температуру, а натомість збільшити тактову частоту інших ядер.

Так як на даний момент існує не дуже багато програм, що використовують для обробки даних все процесорні ядра, особливо, якщо їх більше чотирьох, то застосування технології Turbo Boost дозволяє значно підняти продуктивність процесора, особливо, при роботі з однопоточні додатками.

РОЗДІЛ 3
АНАЛІЗ ТЕХНОЛОГІЇ БАГАТОЯДЕРНОГО ЦЕНТРАЛЬНОГО ПРОЦЕССОРА

3.1 Багатоядерна технологія

Багатоядерна технологія - це звичайно термін, що використовується для опису двох або більше процесорів або ядер, що працюють разом на одній мікросхемі, де один фізичний процесор містить основну логіку двох або більше процесорів.

Ці процесори упаковуються поруч на одному штампі в єдиній інтегральній схемі (IC), але кріплення мікросхеми таке ж, як і в традиційному процесорі. На відміну від багатопроцесорних, багатоядерний є ефективним і підтримує розробку різних типів програмного забезпечення на стандартизованій платформі.

Багатоядерна обробка - це зростаюча галузева тенденція, оскільки одноядерні процесори швидко досягають фізичних меж можливої складності та швидкості. Більшість сучасних систем є багатоядерними. Системи з великою кількістю ядер процесора - десятки або сотні - іноді називають багатоядерними або масово багатоядерними системами.

Технологія багатоядерних процесорів була побудована навколо ідеї можливості зробити паралельні обчислення можливими, оскільки це підвищує продуктивність, швидкість та ефективність роботи комп'ютерів. Наявність декількох ядер на одній мікросхемі мінімізує енерго та теплоспоживання системи, одночасно в змозі значно підвищити загальну продуктивність системи. На рис.3.1 зображено зовнішній вигляд багатоядерного процесора.

Рис.3.1. Багатоядерний процесор [11]

Обсяг продуктивності, отриманий при використанні багатоядерного процесора, залежить від проблеми, що вирішується, використовуваних алгоритмів та їх впровадження в програмне забезпечення (закон Амдала).

Двоядерний процесор з двома ядрами на частоті 2 ГГц може працювати близько до одного ядра 4 ГГц, але кілька ядер забезпечують підвищену обчислювальну здатність на одному чіпі, не вимагаючи складної архітектури.

Як результат, прості багатоядерні процесори мають кращу продуктивність, ніж складні одноядерні процесори. Двоядерні процесори забезпечують два повні ядра виконання, кожен з незалежним інтерфейсом до шини передньої сторони (FSB) і завдяки окремим кешам для кожного ядра операційна система має достатньо ресурсів для паралельного виконання різних завдань, забезпечуючи тим самим значне поліпшення продуктивності.

Ядра в багатоядерному пристрої можуть спільно використовувати один когерентний кеш на найвищому рівні кешу на пристрої, наприклад L2 для Intel Core 2 або можуть мати окремі кеші, наприклад поточні двоядерні процесори AMD, такі як AMD Athlon 64X2.

Оскільки більше обчислювальних навантажень можна виконати паралельно, тому такі виробники, як Intel та AMD намагаються більше зосередитися на обчислювальній роботі та обробці, не збільшуючи тактову частоту

3.2 Багатоядерна архітектура

Багатоядерна архітектура, як будь-яка інша технологія, багатоядерна архітектури різних виробників відрізняються одна від одної. Кількість ядер, конфігурація пам'яті та спільний кеш може відрізнятися. Конструкція, показана на рис.3.2 не є специфічною для будь-якої конкретної багатоядерної конструкції, скоріше, це універсальна конструкція багатоядерної архітектури і не характерна для будь-якого постачальника.

Хоча конструкції виробників відрізняються, все ж є деякі цілісні частини, які входять у кожну багатоядерну архітектуру

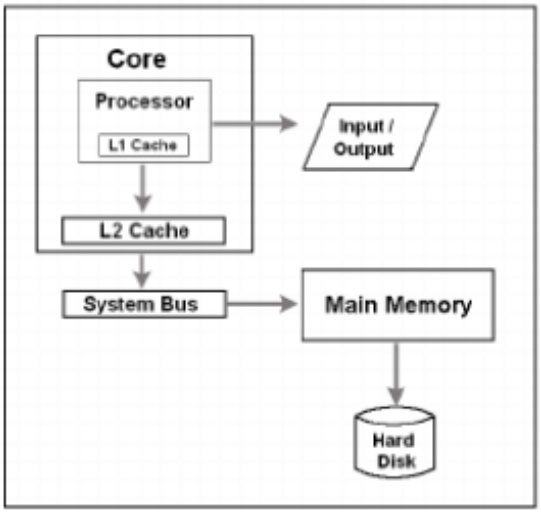

Рис. 3.2. Проста основна архітектура [12]

Процесори працюють, читаючи потік інструкцій і визначають, чи є дані в пам'яті, і операції, що стосуються цього. Звичайні процесори використовували для отримання однієї інструкції з пам'яті та її виконання повністю до запуску некстону.

Але нинішні процесори беруть блоки керуючих сотень чи тисяч інструкцій перед їх виконанням. Вони можуть одночасно виконувати чотири чи більше інструкцій

Кеш рівня 1 або L1 знаходиться найближче до процесора. Це дуже швидка пам'ять для зберігання даних, які часто використовує процесор. Кеш рівня 2 або L2,

хоч і знаходиться поза процесором, і просто повільніше, ніж кеш L1, але все ж набагато швидший, ніж основна пам'ять. Порівняно з кешами L1 та L2, основна пам'ять має дуже великі розміри. Більшість існуючих систем мають пам’ять від 1 ГБ до 16 ГБ при порівнянні кешу приблизно 32 КБ (L1) і 2 МБ (L2).

Коли дані не знаходяться в кеші або основній пам'яті, система отримує їх з жорсткого диска, який є найбільш повільною системою пам'яті. Єдина шина зв'язку під назвою системна шина використовується для зв'язку між ядрами та основною пам'яттю.

3.3 Поточні багатоядерні архітектури та їх порівняння

Intel та AMD є найпопулярнішими виробниками мікропроцесорів. Intel виробляє багато видів багатоядерних процесорів, таких як Pentium D, що використовується у настільних роботах, Core 2 Duo, що використовується як у ноутбуках, так і на робочому столі, та процесор Xeon, який використовується на серверах [13].

AMD має лінійку Althon для настільних ПК, Turion для ноутбуків та Opteron для серверів / робочих станцій. Хоча Core 2 Duo та Athlon 64 X2 працюють на одних платформах, їх архітектура значно відрізняється. Тому проаналізувавши та порівнявши їх архітектурні проекти та те, як ці дві архітектури намагаються вирішити сучасні багатоядерні архітектурні завдання

3.4 Основна архітектура Intel

В основній архітектурі Intel кожне ядро функціонує повністю незалежно від іншого. Вони діляться кешем рівня 2, і завдяки цьому Intel оптимізувала операції для одночасного використання обох ядер. Це означає, що коли обидва ядра працюють на одній області пам’яті, потрібна лише одна копія даних у кеші. Це підвищує ефективність і виділяє більше кешу для інших процесів. Кеш також

динамічно розподіляє свій простір, який повинен використовувати кожне ядро, так що якщо одне ядро не працює, то для іншого ядра буде доступно більше кешу.

Ядро Інтел - це конвеєрна архітектура, де інструкції переміщуються через ряд внутрішніх етапів між входом в процесор і завершенням, як показано на рис.3.4. Після того, як інструкція виходить з етапу, ще один може входити в конвеєр. Ядро має близько чотирнадцяти стадій у своєму трубопроводі, але є ряд ускладнень, таких як дострокове завершення та неврядування, через що важко проаналізувати, скільки саме стадій знаходиться в трубопроводі.

Дует Core 2 Core має перевагу в більш швидкому зв’язку між основними і динамічним спільним кешем між ядрами. Завдяки більш швидкій передачі даних від основного ядра Intel вважається найкращим для багатопотокових програм, які мають велику кількість обміну даними або зв'язку. Але в ситуаціях, коли два ядра мають різні вимоги до кешу, для них це справжнє завдання ефективно керувати ресурсами кешу.

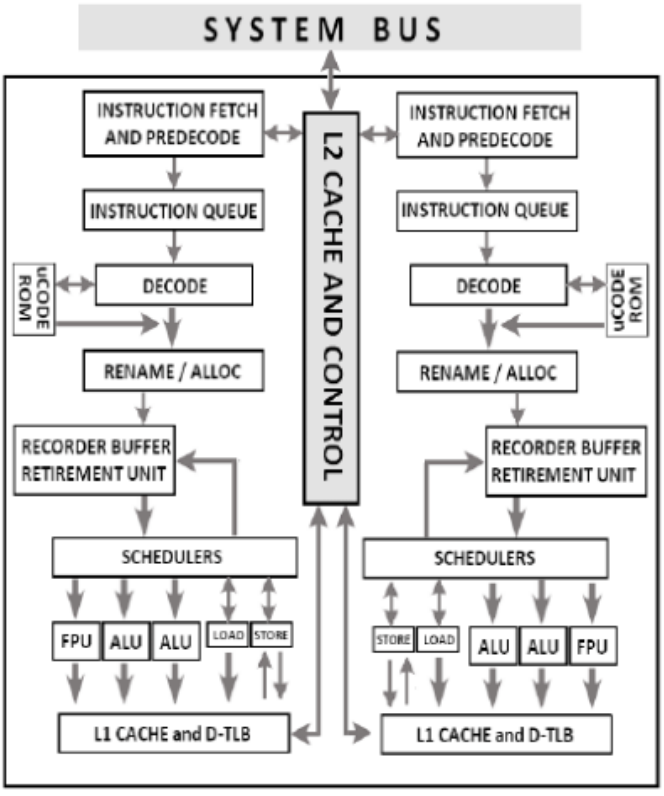

Рис.3.4. Основна архітектура Intel [14]

3.5 Основна архітектура AMD

У 2003 році AMD запустила перший у світі 32-розрядний та 64-бітний процесор, сумісний з архітектурою x86. Процесори AMD усувають вузькі місця, викликані шиною на передній стороні, безпосередньо підключаючи процесор, контролер пам’яті та введення / виведення до центрального процесорного блоку, щоб забезпечити поліпшення загальної продуктивності та ефективності системи.

Процесор AMD Athlon 64X2, розроблений спеціально для багатоядерних архітектур. AMD athlon має приватні кеші L2, як показано на рис.3.5. Обидва ці кеші L2 мають спільний інтерфейс системного запиту, який з'єднується з контролером пам'яті, що визодить з ладу, та HyperTransport.

Інтерфейс системного запиту служить взаємозв'язком між двома ядрами і не вимагає зовнішньої шини. HyperTransport усуває вузькі місця системи, зменшуючи кількість необхідних автобусів. Він забезпечує більшу пропускну здатність, ніж поточна технологія PCI. Останні процесори AMD засновані на архітектурі K8. Деякі приклади - AMD Opteron, Athlon, Turion і Sempron.

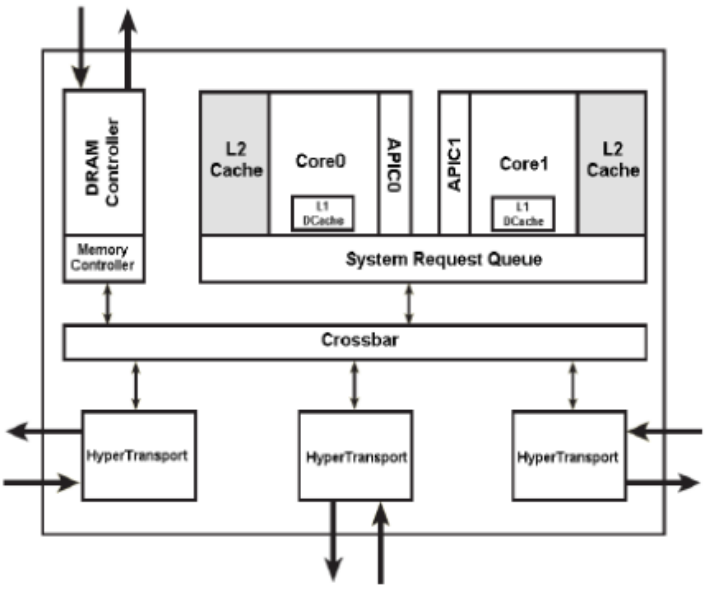

Рис. 3.5. Багатоядерна архітектура AMD [15]

3.6 Порівняння архітектур

На рис. 3.6 показані блок-схеми для Core 2 Duo та Athlon 64 X2 відповідно. Фігура сама пояснює, обидві архітектури є однорідними двоядерними процесорами. Core 2 Duo Intel містить загальну модель пам'яті з приватними кешами L1 та спільним кешем L2.

Коли відбувається помилка кешу L1, кеш L2 та кеш L1 другого ядра проходять паралельно перед відправкою запиту в основну пам'ять. На відміну від цього, Athlon має модель розподіленої пам'яті з дискретними кешами L2. Замість шини, кеш-пам'ять L2 розділяє інтерфейс системного запиту.

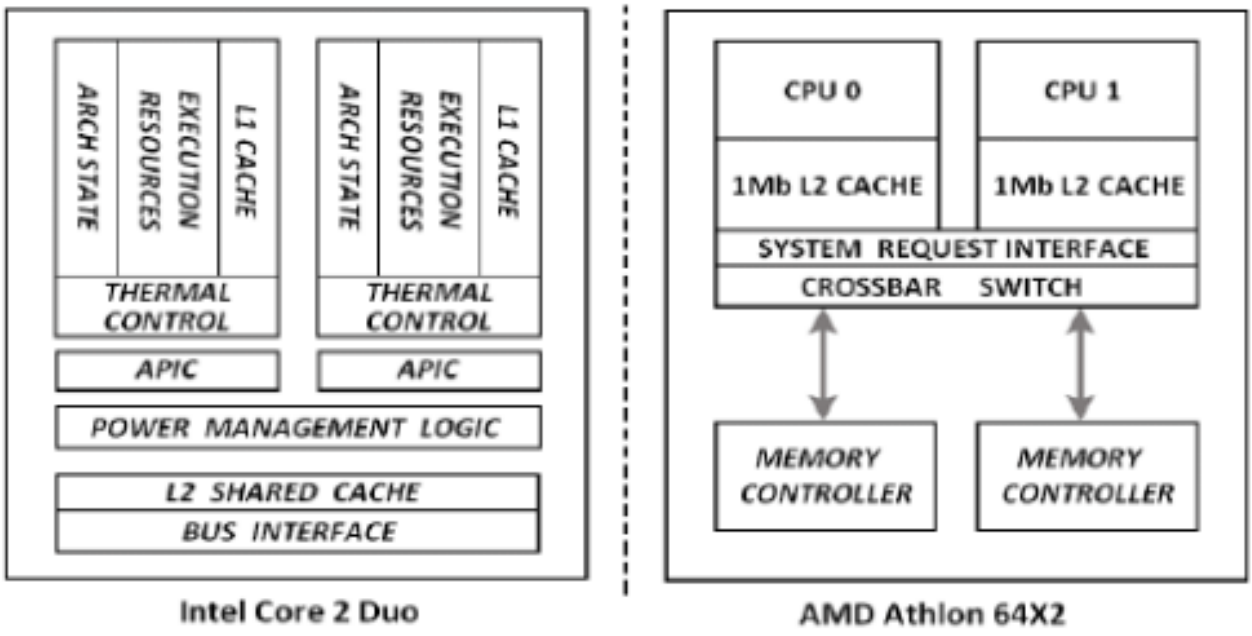

Рис. 3.6. Порівняння архітектури Intel та AMD [16]

3.7 Проблеми при багато ядерності

Збільшування продуктивності, багатоядерного процесори також забезпечують більшу щільність системи та загальну продуктивність.

Багатоядерні процесори використовують менше енергії та генерують менше тепла на ядро, ніж однакова кількість одноядерних процесорів, оскільки вони працюють на менших частотах.

Інновації в дизайні багатоядерних процесорних архітектур приносять нові можливості оптимізації та проблеми для системного програмного забезпечення. Вирішення цих завдань ще більше підвищить продуктивність системи.

Затримка пам’яті була однією з найважливіших вузьких місць в одноядерних процесорах і нині на багатоядерних процесорах [17].

В архітектурі Intel всі ядра мають однакову шину передньої сторони (FSB). Усі дані, що записані або зчитуються з пам'яті, проходять через шину пам'яті. Intel FSB дозволяє до 8 пристроїв проживати в шині.

Оскільки декілька ядер розділяють FSB, тому пропускна здатність буде розділена між усіма ядрами, що є вузьким місцем для продуктивності, оскільки кожне ядро отримує свої інструкції з основної пам'яті.

Оскільки кілька ядер найбільш ефективно використовуються, коли кожне з ядер виконує один потік, отже, одним завданням, що займає пам'ять, можна наситити спільну шину пам'яті, що впливає на загальну продуктивність усіх завдань, що виконуються на цьому процесорі.

AMD архітектура забезпечує своє рішення, маючи власний контролер пам'яті. Однак через фізичний розподіл основної пам’яті виникають деякі інші проблеми, як відображення пам'яті та узгодженість кешу [18].

У сучасних багатоядерних архітектурах Intel використовує спільний кеш L2, тоді як в останній архітектурі AMD використовується індивідуальний кеш L2 для кожного ядра.

Використання високошвидкісного кеш-пам'яті L2 забезпечує важливі переваги для процесорів, такі як збільшення використання простору кешу та більш швидке міжядерне спілкування.

Потенційний недолік використання спільних кешів L2 полягає в тому, що різнорідні шаблони доступу до даних завдань, що займають велику пам’ять,

виконуються на кешах спільного використання ядер, можуть призводити до суперечок кешу і таким чином створювати неоптимальну продуктивність.

Суперечність кешу залежить від спільних ресурсів, кількості активних завдань та послідовності доступу до окремих завдань. Core 2 Duo Intel намагається прискорити узгодженість кешу, маючи змогу одночасно запитувати кеш L1 другого ядра та кешований спільний L2, але спільний кеш L2 у процесорі Core 2 Duo Intel видаляє чіп когерентності кешу на рівні L2 [18].

Розподілений кеш L1 і L2 також може викликати деякі проблеми.

Оскільки кожне ядро має власний кеш, копія даних у цьому кеші може не завжди бути найновішою версією.

Якщо одне ядро записує значення в певне місце, тоді, якщо тим часом друге ядро намагається прочитати це значення з кеша, оновлення копії не буде, оскільки його запис у кеш недійсний і виникає пропуск кешу.

AMDAthlon 64X2 повинен контролювати узгодженість кешу як у кеш-пам'яті L1, так і в L2. AMD використовує технологію взаємозв'язку HyperTransport для швидшого зв'язку між мікросхемами для підтримки узгодженості кешу між двома ядрами. Крім того, Athlon 64X2 має контрольний модуль пам'яті для зменшення затримки доступу до пам'яті [19]

У багатоядерних процесорах кожен пакет процесорів має два або більше виконання ядра, у кожного ядра є свої регістри, блоки виконання, кеші тощо. Швидкі зміни архітектури процесора приносять нові можливості та проблеми системному програмному забезпеченню.

Для отримання оптимальної продуктивності різні компоненти, такі як планувальник завдань, повинні знати про багатоядерну архітектуру та характеристики завдання.

Планувальник процесів керує розподілом ресурсів процесора для завдань і є невід'ємною частиною операційної системи.

Планувальник процесів зазвичай пропонує максимізацію пропускної здатності системи шляхом виконання декількох завдань одночасно та забезпечення справедливості серед запущених завдань у системі.

Можна припустити, що звичайний процес планування для багатоядерних процесорів буде однаковим, але завдяки спільним ресурсам, таким як ієрархія кешу та пропускна здатність пам'яті між ядрами, дозволяє припустити, що планування потрібно робити так, щоб отримати максимальну ефективність.

Теплові проблеми (потужність і температура). Щоб зменшити непотрібне споживання електроенергії, багатоядерна конструкція також повинна використовувати окремий блок управління електроенергією, який може керувати або контролювати непотрібне витрачання енергії.

Щоб це сталося, блок управління живленням повинен вимкнути або вимкнути сердечники, які не спрацьовують часом, або ядра, які часом не потрібні. Також ядра працюють з порівняно нижчою частотою, ніж один процесор, щоб зменшити розсіювання потужності.

Архітектура ядра повинна бути такою, щоб кількість теплоти, виробленої в мікросхемі, була добре розподілена по мікросхемі.

Споживання електроенергії також є функцією кількості транзисторів на мікросхемі. Коли додається більше ядер, щільність транзистора також збільшується, що сприяє споживанню енергії

ВИСНОВКИ

1. Проблеми паралельного програмування пропонуються вирішити за допомогою функціональних мов / гібридів програмування та таких інструментів, як CUDA, RapidMind, openMP, що дозволяють розробникам перетворювати код рефактора.

2. У коротко- та середньостроковій перспективі відбудеться більш широке прийняття в таких областях, як серверні програми (у формі віртуалізації), спеціалізовані програми для обробки (наприклад, візуалізація та висока графіка) та вбудовані програми. Але в основному програмуванні прогрес буде повільним.

3. Незважаючи на інструменти та нові підходи, програмне співтовариство зможе отримати пристойне підвищення продуктивності для додатків.

4. Багатоядерний процесор є важливим доповненням у часовій шкалі мікропроцесорів та засобом вирішення проблем з високим споживанням енергії, ціною та розміром.

5. Виділено архітектуру Intel та AMD, двох найважливіших постачальників сучасних багатоядерних систем, і порівняння зроблено з точки зору архітектури. Аналогічно обговорюються завдання, що стоять перед нами з кеш-пам'яті, пропускної здатності пам’яті та планування планування

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. F. Pollack, "New Microarchitecture Challenges in the Coming Generations of CMOS Process Technologies," in Micro 32, 1999. http://intel.com/research/mrl/Library/micro32Keynote.pdf.

2. R. Merritt, "CPU Designers Debate Multi-core Future," EETimes Online, February2008, ttp://www.eetimes.com/showArticle.jhtml?articleID=206105179.

3. Intel® Core™2 Duo Desktop Processor and Intel® Pentium® Dual Core Processor at http://www.intel.com/products/processor/core2duo/index.htm

4. AMD Athlon™ Processor, AMD Opteron™ Processor and AMD Turion™ 64 X2 Dual-Core at http://www.amd.com/us-en/Processors/

5. AMD, AMD HyperTransport Technology at http://www.amd.com/us-en/Processors/DevelopWithAMD/0,,30_2252_2353,00.html

6. B. Brey, "The Intel Microprocessors," Sixth Edition, Prentice Hall, 2003

7. Lu Peng, Jih-Kwon Peir, Tribuvan K. Prakash, Yen-Kuang Chen and David Koppelman, "Memory Performance and Scalability of Intel's and AMD's Dual-Core Processors: A Case Study," IEEE, 2007.

8. J. Dowdeck, "Inside Intel Core Microarchitecture and Smart Memory Access," Intel, 2006, http://download.intel.com/technology/architecture/sma.pdf

9. Kunle Olukotun, Basem A Nayfeh, Lance Hammond, Ken Wilson and Kunyung Chang, "The case for a single-chip multiprocessor. Architectural Support for Programming Languages and Operating Systems (ASPLOS)," 2006.

10. J. Chang, S. Member, and M. Huang, "The 65-nm 16-mb shared on-die l3 cache for the dual-core intel xeon processor 7100 series," Journal of Solid-State Circuits, 42(4), 2007.

11. L. Peng, J. Peir, T. K. Prakash, Y. Chen, and D. Koppelman, "Memory performance and scalability of intel's and amd's dual-core processors: A case study," in International Performance, Computing, and Communications Conference (IPCCC), 2007.

12. E. Suh, L. Rudolph, and S. Devadas, "Dynamic partitioning of shared cache memory," The Journal of Supercomputing, vol. 28, no. 1, April 2004, pp. 7–26.

13. R. Iyer, "CQoS: A framework for enabling QoS in shared caches of CMP platforms," in ICS, 2004.

14. S. Kim, D. Chandra, and Y. Solihin, "Fair cache sharing and partitioning in a chip multiprocessor architecture," in PACT, 2004.

15. M. Rajagopalan, B. T. Lewis and T. A. Anderson: "Thread Scheduling for Multi-Core Platforms," in proceedings of the Eleventh Workshop on Hot Topics in Operating Systems, 2007

16.Venu, Balaji. "Multicore processors- An overview."

17. Jr., Ransford Hyman. "Performance issues on Multicore Processors"

18. Barbic, Jernej. "Multicore architectures." 2007.

19. Martin, Christian. "Multicore Processors: Challenges, Opportunities, emerging Trends." Embedded World 2014 Exhibition and Conference . 2014.