Комп'ютерні системи.

Для студентів напрямку підготовки
«Комп’ютерні науки» та «Комп’ютерна інженерія»

ЗМІСТ

1. Введення.

Для виконання обчислень, необхідність яких виникає в науковій, інженерній та виробничій діяльності, широко використовуються електронні обчислювальні машини та системи: ЕОМ, комп'ютери, комп'ютерні системи (КС).

ЕОМ – це комплекс апаратури, що забезпечує виконання набору операцій над інформацією, що є достатнім для реалізації деякого класу алгоритмів. Реалізація довільного алгоритму в КС потребує виконання певної послідовності машинних операцій, для породження якої використовується *програма* – алгоритм перетворення інформації в термінах команд. Команди ініціюють операції, що виконуються апаратурою ЕОМ. Таким чином, кожне застосування КС потребує комплектацію машини набором програм, які несуть в собі інформацію про те, що повинна виконувати система.

Система, що складається з однієї або декількох ЕОМ та набору програм, що забезпечують виконання покладених на систему функцій, називається *комп'ютерною системою*. Узагальнено склад КС зображено на рис.1.1.

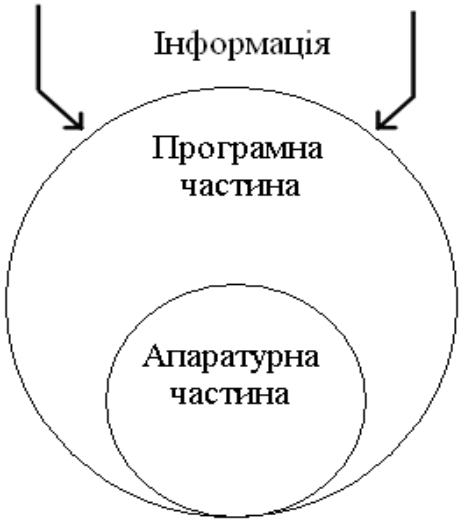


Рис.1.1. Склад КС.

Апаратурна частина КС містить в собі сукупність пристроїв, що входять до складу однієї ЕОМ або цифрового обчислювального комплексу, який складається з декількох пов'язаних ЕОМ. Призначення апаратурної частини ЕОМ – виконання операцій введення, збереження та виводу інформації. Програмна частина КС – набір програм, що задає порядок реалізації покладених на систему функцій в термінах команд, які реалізуються апаратурною частиною системи. Програми керують роботою апаратури з метою отримання результатів, що

відповідають призначенню системи. КС взаємодіє з зовнішньою середою, отримуючи від неї необхідну інформацію, та виводячи інформацію, що нею вироблена.

КС як об’єкт характеризується:

- множиною даних, що вводяться у систему, та множиною даних, що формуються на її виході;
- функцією системи, що встановлює правило відображення множини вихідних даних в множину результатів;
- структурою системи, що визначає вхід та вихід системи, номенклатуру пристроїв та зв’язки між ними, номенклатуру алгоритмів та порядок їх взаємодії між собою та апаратурою системи.

Програмне забезпечення включає в себе прикладні та керуючі програми. *Прикладні програми* є алгоритмами виконання функцій, реалізація яких покладена на систему. Між програмами існують інформаційні зв’язки, які виникають у випадку, коли результати роботи однієї програми є вхідними даними для іншої. Крім того, для виконання програм використовується спеціальний набір пристроїв, що по черзі розподіляється між програмами. Тому необхідно передбачити механізм керування порядком виконання програм, для чого використовується набір керуючих програм. *Керуючі програми* забезпечують необхідний порядок взаємодії прикладних програм між собою та з обладнанням КС. Порядок розподілу обчислювальних процесів в просторі та часі визначається *стратегією керування обчислювальними процесами* в КС. Стратегія керування задає алгоритм планування робіт в системі, який реалізують керуючі програми. Таким чином, внутрішню організацію КС можна розглядати як сукупність трьох основних компонент: прикладних програм, керуючих програм та обладнання (рис.1.2).

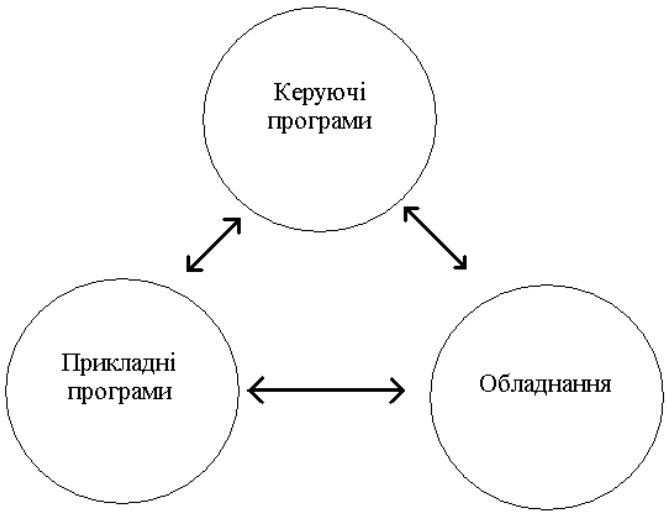


Рис.1.2. Компоненти КС.

При створенні КС необхідно, виходячи із відомостей щодо призначення системи, визначити її структуру та номенклатуру алгоритмів, які найкращим чином відповідають призначенню системи. Щоб вирішувати такого роду задачі необхідно знати: як впливають

різноманітні способи структурної організації КС та керування обчислювальними процесами на характер протікання останніх в КС. Властивості та закономірності, що притаманні обчислювальним процесам, які проходять в системах з різною організацією, складають предмет теорії КС.

Теоретичні дослідження базуються на використанні моделей реальних об'єктів. Моделі відображають об'єкт дослідження в формі, що є необхідною та достатньою для отримання результатів, які складають ціль дослідження. Моделі, що досліджуються, є компромісом між двома суперечливими тенденціями. З однієї сторони бажано, щоб модель якнайповніше відображала в собі реальні процеси; з іншої модель повинна бути достатньо простою, щоб можна було отримати бажані результати. В теорії КС обчислювальні процеси вивчаються з таким ступенем деталізації, який можна отримати, якщо окремі пристрої ЕОМ розглядати як елементи системи, а алгоритми – як сукупність операторів, кожний з яких розглядається як об'єкт, що породжує визначену дію: введення, передачу, перетворення або виведення інформації. Іншими словами, в моделях, що вивчаються в теорії КС, алгоритм є сукупністю запитів до технічних ресурсів – пам'яті, засобам обробки, введення та виведення інформації, а обладнання ЕОМ – як сукупність об'єктів, що обслуговують ці запити. Такий ступінь деталізації систем в моделях є достатнім для визначення наступних важливих характеристик систем: продуктивність, час виконання роботи, завантаження окремих пристроїв, інтенсивності потоків інформації і таке інше.

Теорію КС спрямовано на розробку методів синтезу оптимальних структур ЕОМ та стратегій керування обчислювальними процесами, причому структури виявляються з точністю до приладів. Синтез приладів є предметом теорії ЕОМ. Розробка програм, що реалізують прикладні алгоритми та стратегії керування обчислювальними процесами, проводиться з використанням методів теорії алгоритмів та апарату програмування.

1.1. Загальні принципи дії ЕОМ.

Спрощену структуру ЕОМ наведено на рис.1.3. ЕОМ містить натупні загальні складові: арифметико-логічний пристрій, пам'ять, керуючий пристрій, пристрій введення даних в машину, пристрій виведення результатів розрахунків та, зазвичай, термінал керування.

Арифметико-логічний пристрій (АЛП) робить арифметичні і логічні перетворення над машинними словами, що надходять до нього, тобто кодами визначеної довжини, що

представляють собою числа або інший вид інформації. Кількість розрядів у машинному слові зазвичай збігається з розрядом в основних регістрах АЛП.

Пам'ять зберігає інформацію, передану з інших пристроїв, у тому числі, що надходить у машину зовні через пристрої введення, і видає в усі інші пристрої інформацію, необхідну для протікання обчислювального процесу. Пам'ять машини в більшості випадків складається з двох частин: швидкодіючої *основної* або *оперативної* (внутрішньої) *пам'яті* (ОП) та більш повільної, але здатної зберігати значно більший обсяг інформації *зовнішньої пам'яті*.

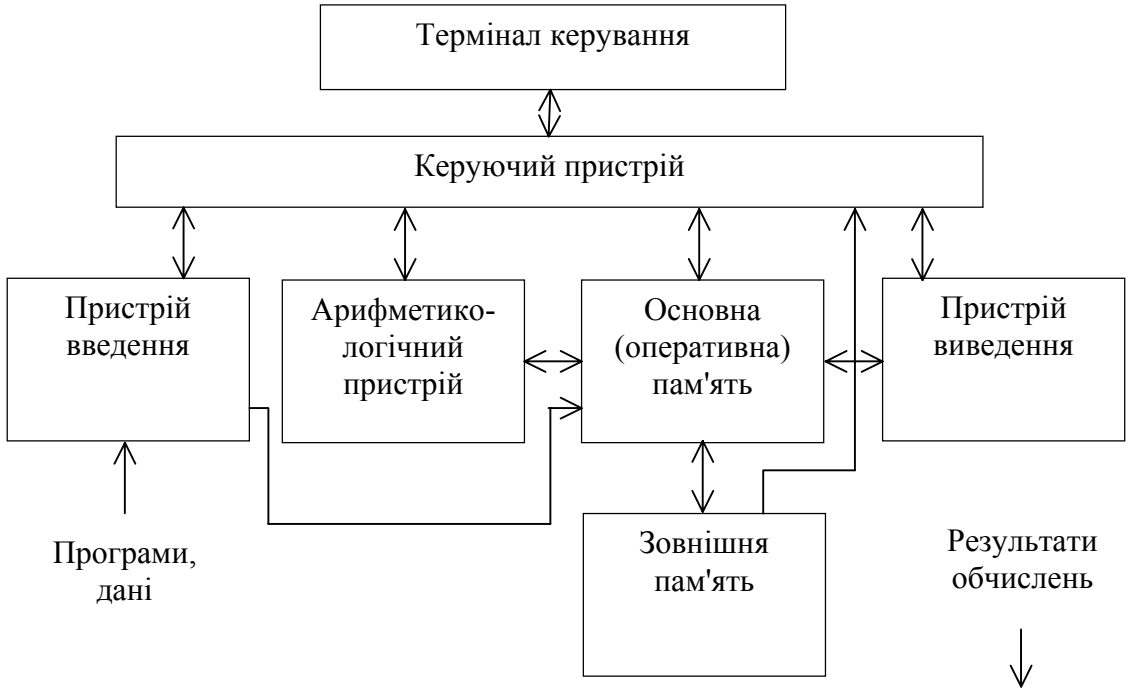


Рис.1.3. Структура електронної обчислювальної машини

Оперативна пам'ять містить деяке число комірок, кожна з яких служить для збереження машинного слова або його частини. Комірки нумеруються, номер комірки називається її *адресою*.

У *запам'ятовуючих пристроях*, що реалізують в ЕОМ функцію пам'яті, виконуються операції читання збереженої інформації для передачі в інші пристрої та запису інформації, що надходить з інших пристроїв. При читанні слова з комірки вміст останньої не змінюється і при необхідності може бути знову узятий з тієї ж комірки. Безпосередньо в обчислювальному процесі бере участь тільки ОП.

Керуючий пристрій (КП) автоматично без участі людини керує обчислювальним процесом, посилаючи всім іншим пристроям сигнали, що наказують виконати ті чи інші дії, зокрема включає АЛП на виконання деякої операції.

Іншим найважливішим принципом є принцип, за яким програми зберігаються в пам'яті. Згідно з цим принципом команди, що закодовані в цифровому вигляді, зберігаються в пам'яті нарівні з числами. У команді вказуються не самі числа, які беруть участь в операціях, а адреси осередків ОП, в яких вони знаходяться, та адреса комірки, куди поміщається результат операції. Таким чином, над командами як і над числами машина може виробляти деякі операції.

Розглянута архітектура називається фон-неймановською і запропонована американським математиком Джоном фон Нейманом в 1945 році. Вона знайшла широке застосування завдяки своїй простоті і великий гнучкості при управлінні обчислювальним процесом і донині домінує при побудові різних ЕОМ.

Більшість сучасних КС працюють за принципами, які сформульовано Джоном фон Нейманом, і тому відносяться до типу фон-неймановських. До цих принципів відносять:

Принцип програмного керування. Програма складається з набору команд, які виконуються автоматично процесором одна за іншою за встановленим порядком. Після виконання деякої команди лічильник команд (ЛК) збільшує адрес наступної команди. Є команди (безумовного та умовного переходів), які можуть змінювати адрес команди в ЛК.

Принцип однорідності пам'яті. Програми та дані зберігаються в тій самій пам'яті. Процесор не розрізняє, що саме зберігається в комірці пам'яті, тому над командами можна виконувати ті самі дії, що і над даними. Тому програма під час виконання може змінювати сама себе.

Принцип адресності. Основна пам'ять складається з нумерованих комірок та процесору у довільний момент часу доступна будь-яка комірка пам'яті. Тому області пам'яті можна іменувати та звертатися до них за цим ім'ям.

Формулювання принципів побудови електронних обчислювальних машин належить групі математиків Принстонського університету, до якої входили Дж. Нейман, Г. Голдстайн та А. Беркс. Після успішного завершення роботи над проектом "Використання швидкодіючих електронних пристроїв для обчислень", результатом якого була побудова першої ЕОМ ENIAC, постала задача проаналізувати сильні та слабкі сторони проекту та дати відповідні рекомендації. Рішення цієї задачі було дано у звіті "Попереднє обговорення логічного конструювання електронного обчислювального пристрою" вже згаданих авторів. Саме ідеї, які

було сформульовано у звіті, зробили серйозний вплив на розвиток обчислювальної техніки. Після публікації цього звіту почалися наукові дослідження в галузі електронних ОМ.

Ось як було сформульовано принципи у згаданому звіті.

1) Машини на електронних елементах повинні працювати не в десятинній, а у двійковій системі счислення.

2) Програма повинна розміщуватися в одному з блоків машини – у запам'ятовуючому пристрої, який повинен мати належну ємність та відповідати швидкостям вибірки команд та запису даних.

3) Програма так само як числа записується у двійковому коді. Таким чином за формою представлення команди та дані однотипні, що веде до наступних наслідків:

- проміжні дані обчислень, константи та інші числа можуть розміщуватися в тому самому пристрої, що і програма;
- числова форма запису програми дозволяє машині проводити операції над величинами, якими закодовано команди програми.

Труднощі фізичної реалізації запам'ятовуючого пристрою, швидкодія якого відповідає швидкості роботи логічних схем, потребує ієрархічної організації пам'яті.

Арифметичний пристрій машини конструюється на базі схем, які виконують операції додавання; побудова спеціалізованих пристроїв для виконання інших операціє є недоречною.

В машині використовується паралельний принцип організації обчислювального процесу: операції над словами виконуються одночасно за всіма розрядами.

Заслуга фон Неймана полягає не тільки в тому, що він сформулював дані принципи (наприклад, в роботах Бебіджа передбачався запам'ятовуючий пристрій для зберігання чисел, Вінер у 1940 році показав доцільність використання обчислювань у двійковому коді, а в проектах Цузе та Атанасова передбачалося виконання операцій у двійковому коді), але й і в тому, що він розробив методи їх практичної реалізації: проекти EDVAC та IAC. Перший звіт Неймана містив характеристику проекту EDVAC, другий – детальну характеристику вузлів (пристрою керування, введення-виведення, запам'ятовуючого та арифметичного пристроїв) обчислювальної машини IAC. Наступний досвід розробки ЕОМ виявив правильність висновків Неймана, за виключенням рекомендації використовувати програмні, а не схемні методи для представлення чисел в формі з плаваючою комою.

Слід зазначити, що в Радянському Союзі, а саме в Україні, незалежно від фон Неймана було сформульовано аналогічні принципи. При цьому деякі дослідники вважають, що це було зроблено навіть раніше. Під час ІІ світової війни Сергій Олександрович Лєбєдєв (1902-1974) працював над вдосконаленням зброї та прийшов до висновку необхідності автоматизації

швидких обчислень, тобто розробки електронних обчислювальних машин, та почав ескізно опрацьовувати концепцію. Його працею зацікавилися в Московському енергетичному інституті, який перший на теренах Радянського Союзу почав випускати фахівців з обчислювальної техніки. Для обговорення питання було організовано зустріч Лєбєдєва в ЦК комуністичної партії. Після доповіді Лєбєдєва, в якій було озвучено швидкодію майбутньої ЕОМ на рівні 1000 операцій за секунду, висновок був приголомшливий:"Що ж це, ми за один-два місяці вирішимо усі задачі, а що потім – ЕОМ на смітник?". Після невдачі у Москві, роботою Лєбєдєва зацікавилися в Україні. Було організовано його зустріч з Президентом АН УРСР О.О.Богомольцем, який запропонував Лєбєдєву балотуватися в академіки та посаду директора Інституту енергетики АН України. Після призначення Лєбєдєва директором цього інституту, в 1947 році було організовано лабораторію, перед якою стояло завдання якнайшвидше розробити та впровадити в експлуатацію ЕОМ. В кінці 1950 року мала електронно-обчислювальна машина (МЕОМ) почала працювати, та її було прийнято в експлуатацію Державною комісією на чолі з майбутнім президентом АН СРСР М.В. Келдишем. На той момент це була єдина на континентальній Європі діюча ЕОМ.

На відміну від зарубіжних машин МЕОМ використовувала не двійково-десятичні принципи рахування, а винятково двійкову систему обчислення, не відокремлену пам'ять на штекерах, а загальну оперативну пам'ять для даних та програм. Ці принципи в майбутньому були широковживаними при проектуванні ЕОМ.

Ось як сформулював Лєбєдєв принципи побудови ЕОМ:

- в склад ЕОМ повинен містити пристрої арифметики, пам'яті, введення-виведення інформації, керування.
- програма обчислень кодується та зберігається у пам'яті подібно до чисел;
- обчислення повинні виконуватися автоматично на базі програми, що зберігається в пам'яті;
- крім арифметичних вводяться логічні операції: порівняння, умовного та безумовного переходів, кон'юнкція, диз'юнкція та заперечення;
- пам'ять слід будувати за ієрархічним принципом;
- для обчислень використовуються обчислювальні методи рішення задач.

Під керівництвом С.О. Лєбєдєва в подальшому були сконструйовані лампові ЕОМ ВЕОМ-2 та М-20, напівпровідникові варіанти М-20: М-220 та М-222, а також ВЕОМ-3М та ВЕОМ-4.

1.2.Еволюція обчислювальних систем.

Зупинимося на деяких тенденціях розвитку обчислювальних систем останнього часу.

1) *Поділ мікрокомп'ютерів на два класи: персональні комп'ютери і робочі станції.* Передумови для цього створюють підвищення обчислювальної потужності мікропроцесорів, розвиток технологій інтегральних схем, упровадження «дружніх інтерфейсів». Слід також зазначити, що сучасні високопродуктивні ПК впритул за характеристиками наближаються до робочих станцій; з іншої сторони робочі станції мають повні набори офісних додатків. Таким чином, провести точну границю між класами цих ЕОМ складно.

2) *Розукрупнення (downsizing).* Обидва типи (і персональні комп'ютери і робочі станції) останнім часом розглядаються як потенційний ресурс для мереж масштабу підприємства. У результаті практично пішли зі сцени старомодні мінікомп’ютери з їх патентованою архітектурою і використанням терміналів, що приєднуються до головної машини. По мірі розвитку цього процесу і збільшення продуктивності платформи Intel найбільш могутні ПК (але все-таки частіше відкриті системи на базі UNIX) стали використовуватися як сервери, поступово замінюючи мінікомп’ютери. Останнім часом процес сповільнився, що зв'язано зі складністю переходу на розподілені обчислення, зокрема до моделі «клієнт-сервер».

3) *Розвиток багатопроцесорних ОС*, що обумовлено зростаючими вимогами до обчислювальної потужності комп'ютерів у всіх сферах їхнього застосування.

4) *Розвиток систем з розподіленими обчисленнями* таких як архітектура «клієнт-сервер», паралельні бази даних і т.і.

5) *Розвиток кластерних архітектур*. Дана тенденція обумовлена необхідністю побудови обчислювальних систем для критично важливих застосувань, зв'язаних з обробкою транзакцій, керуванням базами даних і обслуговуванням телекомунікацій, для чого необхідне забезпечення високої продуктивності і тривалого функціонування систем. Найбільш ефективний спосіб досягнення заданого рівня продуктивності - застосування архітектур, що здатні до масштабування. Для систем даного типу застосовується термін «системи високої надійності» (High Availability Systems).

6) *Розвиток обчислювальних мереж* як локальних, так і глобальних (Internet). Широко відомі випадки застосування потужності комп'ютерів, що мають доступ до мережі Internet, для рішення складних задач, зокрема для злому алгоритмів, що шифрують.

1.3. Класифікація КС.

Мета класифікації – розділити безліч можливих систем на класи подібних систем для вивчення властивостей окремих класів. Найбільш результативною вважається така класифікація, що приводить до максимальної уніфікації теоретичних досліджень і прикладних методів. За основну ознаку при класифікації виберемо призначення системи (рис.1.4).

1.3.1. Класифікація за призначенням системи.

Проблемно-орієнтовані КС призначені для виконання фіксованого набору алгоритмів, тобто для рішення конкретних задач. Приклади проблемно-орієнтованих КС: системи керування технологічними процесами, резервування і продажі квитків, обробки банківських документів і т.п. Задачі й алгоритми їхнього рішення відомі на момент створення КС і не змінюються протягом усього періоду експлуатації. Проблемно-орієнтовані системи розділяються на цифрові керуючі системи та системи типу «запит-відповідь».

Призначення *цифрової керуючої системи* – керування роботою конкретного об'єкта. Набір програм визначається задачею керування і незмінний під час роботи системи. Програми в таких системах повинні виконуватися в темпі, обумовленому динамікою керуючого об'єкта.

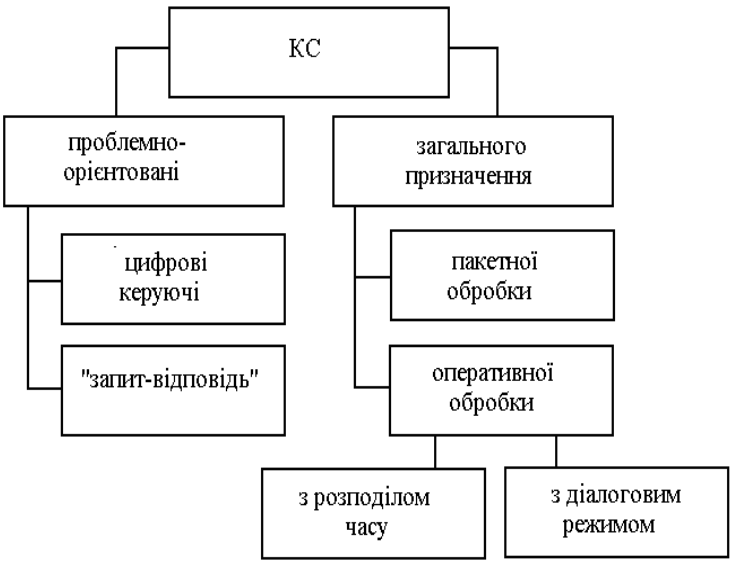


Рис.1.4. Класифікація КС за призначенням.

Наднормативна затримка може привести до катастрофічних наслідків, або до зниження показників роботи керованого об'єкта.

Системи типу *«запит-відповідь»* призначені для обслуговування не технічних об'єктів, а людей. Запит, що формується на вході системи, є вимогою ініціювати визначену програму, що формує відповідь. Система повинна обслуговувати запити таким чином, щоб мінімізувати середній час запитів, тоді як у керуючих системах необхідно виконувати чіткі вимоги на час одержання результатів.

Загальним для цих двох типів пристроїв є та властивість, що темп виконання програм підкоряється темпу процесів, що розвиваються поза системою, тобто підкоряється ходу реального часу. Такий режим роботи КС називається роботою в *реальному масштабі часу* (РМЧ).

КС загального призначення вирішують задачі, номенклатура яких не піддається строгому визначенню на момент створення системи. Найчастіше це задачі, що виникають у науковій, виробничій і оборонній діяльності, та називаються прикладними задачами, а програми їхнього рішення – *прикладними програмами*.

Один із способів організації обчислення – максимальне число задач, розв'язуваних в одиницю часу. Для цього в розпорядження КС необхідно представити за можливістю більший набір робіт. КС, виконання робіт у яких організується шляхом завантаження в систему пакета (набору) задач, що оброблюються у порядку, орієнтованому на мінімізацію часу обробки всього пакету, називаються *системами пакетної обробки* (СПО). Однак при такому способі організації час обробки кожної задачі окремо виявляється далеко не мінімальним.

У ряді випадків обчислення доцільно організовувати таким чином, щоб мінімізувати середній час рішення задач: проміжок часу від моменту надходження задачі на обробку до моменту її закінчення. КС загального призначення, використовувані для рішення задач у режимі, що забезпечує мінімальний середній час одержання результатів по кожній задачі, називається *системою оперативної обробки* (СОО). Для мінімізації середнього часу рішення задач система повинна мати наступні властивості:

- приймати задачу на обробку в момент її виникнення;
- обслуговувати одночасно декілька користувачів, забезпечуючи для них паралельне введення-виведення і, за можливістю, обробку програм;
- виконувати задачі в порядку, що забезпечує мінімум середнього часу перебування задач у системі.

Розрізняють два способи організації оперативної обробки: розподіл часу та діалоговий режим.

Спосіб розподілу часу складається в почерговому виділенні кожному користувачу кванта процесорного часу, у результаті чого середній час відповіді для окремого користувача зменшується. Такі КС називаються *системами з розподілом часу* (СРЧ).

При іншому способі взаємодія користувача і системи організується таким чином, щоб процес рішення задачі був розбитий на досить невеликі етапи, що дозволяє ЕОМ обслуговувати велике число користувачів одночасно. КС, призначені для рішення задач, представлених у вигляді сукупності незалежних кроків, називаються *системами з діалоговим режимом роботи*.

1.3.2. Класифікація за швидкодією

При такій класифікації продуктивність КС характеризується побічно – за сумарною швидкодією процесорів, що входять до системи. Швидкодія процесора, зазвичай, визначається кількістю елементарних операцій, що реалізуються в одиницю часу. При цьому необхідно врахувати ймовірність використання різноманітних операцій.

Проте до теперішнього часу не виявлено міру «обчислювальної роботи», яка придатна для практичних вживань. Проте практика змушує порівнювати системи за даною ознакою, що породило ряд тестів. Розглянемо основні з них, які використовуються на практиці.

MIPS – мільйон операцій в секунду, тобто для будь-якої даної програми MIPS є просто відношення кількості команд в програмі до часу її виконання. Таким чином, продуктивність може бути визначена як зворотна до часу виконання величина, причому більш швидкі машини матимуть вищий рейтинг MIPS.

Недоліки:

- залежить від типу команд процесора;
- змінюється від програми до програми навіть на одному комп'ютері;
- може змінюватися по відношенню до продуктивності у зворотній бік. Приклад: при заміні підпрограм роботи з числами з плаваючою крапкою на роботу арифметичного співпроцесора швидкодія системи зростає (розрахунки є швидшими), тоді як індекс MIPS падає (використовуються повільніші команди процесору).

MFLOPS – мільйон елементарних арифметичних операцій над числами з плаваючою крапкою, виконаних в секунду. Застосовується лише для оцінки операцій з плаваючою крапкою. Він базується на кількості виконуваних операцій, а не на кількості виконуваних команд. Тобто, одна і та ж програма, що працює на різних комп'ютерах, виконуватиме різну

кількість команд, але одну й ту ж кількість операцій з плаваючою крапкою. Саме тому рейтинг MFLOPS призначався для справедливого порівняння різних машин між собою.

Недоліки:

- набори операцій з плаваючою крапкою несумісні на різних комп'ютерах: наприклад, комп'ютери Cray замість ділення використовують здобуття зворотної величини і множення;
- індекс змінюється від типа операцій, використовуваних в суміші команд: буде вище для суміші із 100% команд складання і нижче для 100% команд ділення.

1.3.3. Класифікація за структурою КС.

Залежно від складу апаратурної частини виділяють:

1) *Одномашинна КС*: включає одну ЕОМ, що містить єдиний процесор і необхідний комплект супутніх пристроїв. Така КС здатна виконувати розрахунки за однією програмою в кожен момент часу і, можливо, декілька операцій введення-виведення.

2) *Мультипроцесорна КС*: містить декілька процесорів, здатних паралельно та незалежно обробляти програми. Процесори зв'язуються з великою кількістю периферійних пристроїв і, можливо, із загальною оперативною пам'яттю. Мультипроцесорна система називається однорідною, якщо вона містить однакові процесори, інакше – неоднорідною.

3) *Багатомашинна КС*: складається з декількох зв'язаних між собою ЕОМ, які розташовані в безпосередній близькості. Кожна ЕОМ містить свій процесор і набір периферійних пристроїв і працює велику частину часу самостійно, звертаючись до інших ЕОМ з метою передачі інформації. Багатомашинні КС також розділяються на однорідні та неоднорідні залежно від однотипності або різнотипності ЕОМ, що входять до їхнього складу. КС з декількох територіально розділених ЕОМ, зв'язаних між собою каналами передачі даних, називається *обчислювальною мережею*.

1.3.4. Функціональна класифікація комп'ютерів.

Однією з знак, за якою класифікуються системи, є сфера їх застосування. Це найбільш розмита схема, тому що, зазвичай, існує декілька приблизно рівноцінних варіантів вибору моделей систем для рішення поставленого кола задач. В той же час є сфери переважного

використання моделей тих або інших класів. Вони-то і складають деяку стійку основу для функціональної класифікації. В даний час прийнято виділяти наступні групи: суперкомп'ютери, універсальні комп'ютери, міні-комп'ютери, персональні комп'ютери і вбудовуванні процесори.

Суперкомп'ютер, або супер-ЕОМ, є самостійною надпотужною та зазвичай багатопроцесорною обчислювальною системою, яка здатна вирішувати завдання граничних класів, тобто такі завдання, в яких доводиться з максимально можливими швидкостями обробляти величезні масиви даних. До них відносяться, наприклад, завдання метеопрогнозу в планетарних масштабах, управління системами протиракетної і космічної оборони, розрахунку і проектування сучасних літаків і космічних кораблів, завдання з області ядерної фізики і досліджень космогонічних теорій, завдання забезпечення роботи глобальних мереж загальносвітового значення і таке інше. Для їх вирішення і досягнення необхідного для цього рівня продуктивності суперкомп'ютери містять десятки тисяч процесорів. Вартість суперкомп'ютерів може доходити до 500 млн. доларів і більше.

Станом на літо 2005 р. щонайпотужнішим суперкомп'ютером в світі була американська обчислювальна система Blue Gene/L, яка містить 65 536 процесорів. Пікова продуктивність цієї системи дорівнювала 136,8 Тфлоп (136,8 трлн. операцій з плаваючою крапкою в секунду). Відзначимо, що проектна швидкість цього суперкомп'ютера 360 Тфлоп. Російська машина МВС-1500/ВМ, що складається з 924 процесорів, з піковою продуктивністю 8,1 Тфлоп в списку щонайпотужніших машин світу в цей же час займала непогане 56-е місце. Відмітимо, що в цей час у всьому світі було лише дев'ять суперкомп'ютерів, максимальна швидкодія яких перевищувала 10 Тфлоп. В даний час вже проектуються системи, швидкість роботи яких вимірюватиметься пета-флопами — квадриліонами флопів.

Група *універсальних комп'ютерів*, або мейнфреймів (від mainframe — головний каркас, центральна будова), характеризується можливістю вирішувати переважну більшість завдань обробки інформації і практично необмеженими, можливостями її зберігання. Універсальні машини є великими обчислювальними системами, що містять, зазвичай, один або невелику кількість процесорів. Вони використовуються як центральна ланка в системах управління виробничим циклом, в обчислювальних центрах крупних підприємств, вищих учбових закладів, дослідницьких центрів, а також як масові сховища інформації. Останнім часом універсальні комп'ютери часто застосовуються як провідний елемент глобальних і локальних мереж, який надає свої обчислювальні ресурси всім підключеним до мережі комп'ютерам. До групи універсальних комп'ютерів відносять машини типу ЄС ЕОМ. Ця група машин поступово витісняється потужними персональними комп'ютерами.

Міні-комп'ютери, або міні-ЕОМ, використовувалися для роботи в умовах реального виробництва для управління потоковими лініями, як центральна ланка в системах управління устаткуванням цеху або відділу у великих організаціях, а також для забезпечення роботи середніх за розмірами організацій. Міні-комп'ютери були дешевшим варіантом універсальних машин. Як правило, міні-комп'ютери виконувалися у вигляді декількох підлогових стійок, що містять всі пристрої. В даний час ця група машин вважається застарілою, вони практично повністю витиснені потужнішими і дешевшими персональними комп'ютерами.

Персональні комп'ютери (застарілі назви — мікрокомп'ютери, мікро-ЕОМ) — це група машин настільного виконання, які експлуатуються, як правило, однією людиною або невеликим колективом фахівців для вирішення своїх професійних завдань. Інколи персональний комп'ютер використовується як провідний елемент системи управління групою механізмів.

За своїми обчислювальними можливостями сучасні персональні комп'ютери залишили далеко позаду себе машини другого і третього поколінь, не говорячи вже про машини першого покоління. Для більшої наочності можливо порівняти тридцяти тонний «динозавр» ENIAC з його розмірами і швидкістю в 5 тис. операцій в секунду і стандартний сучасний персональний комп'ютер, що уміщається на робочому столі фахівця і виконує мільярди операцій в секунду.

Вбудовані процесори є програмованими інтегральними схемами, що включаються в конструкцію якого-небудь окремого пристрою або механізму (автомобіль, металоріжучий верстат, крилата ракета) з метою автоматизації управління або оптимізації його роботи. Якщо така мікросхема з'єднується з якими-небудь пристроями, що запам'ятовують, і пристроями обміну даними, то таку конструкцію називають вбудованим комп'ютером. Виявляється, що вигідно вбудовувати в різні пристрої і механізми по-різному запрограмовані, але однотипні процесори або комп'ютери, чим для кожного з них заново розробляти унікальні пристрої управління.

Існує ще одна класифікація, заснована на мережевій моделі *«клієнт-сервер»*. Комп'ютери, які надають свої ресурси іншим комп'ютерам, прийнято називати серверами (від serve — обслуговувати, бути корисним), а машини, які ці ресурси використовують, — клієнтами. З цієї точки зору в групу серверів потрапляють підключені до мережі суперкомп'ютери, універсальні комп'ютери і потужні персональні машини, які в цьому випадку називаються робочими станціями. До групи клієнтів входять персональні комп'ютери, що грають роль так званого інтелектуального терміналу, - потужнішого, ніж звичайний термінал, пристрою, який не лише забезпечує обмін даними або управління роботою комп'ютера в мережі, але і може узяти на себе значну частину функцій по зберіганню і обробці

інформації. Крім того, до групи клієнтів відносяться так звані без дискові робочі станції — персональні машини без зовнішньої пам'яті, а також термінальні установки — крайові пристрої, що складаються з клавіатури і дисплея і, зазвичай, не мають власного процесора і пам'яті.

Серед клієнтів досить часто виділяють X-термінали, які є комбінацією бездискових робочих станцій і стандартних ASCII-терміналів. Типовий X-термінал включає наступні елементи:

- екран високої роздільної здатності, зазвичай розміром від 14 до 21 дюйма по діагоналі;
- мікропроцесор на базі Motorola 68xxx або RISC-процесор типа Intel i960, MIPS R3000 або AMD29000;
- окремий графічний співпроцесор на додаток до основного процесора, що підтримує двопроцесорну архітектуру, яка забезпечує швидше малювання на екрані і прокручування екрану;
- базові системні програми, на яких працює система X-Windows і виконуються мережеві протоколи;
- програмне забезпечення сервера X11;
- змінний об'єм локальної пам'яті (від 2 до 8 Мбайт) для дисплея, мережевого інтерфейсу, що підтримує TCP/IP і інші мережеві протоколи;
- порти для підключення клавіатури і миші.

X-термінали відрізняються від ПК і робочих станцій не лише тим, що не виконують функції звичайної локальної обробки. Робота X-терміналів залежить від головної (хост) системи, до якої вони підключені за допомогою мережі. Для того, щоб X-термінал міг працювати, користувачі повинні інсталювати програмне забезпечення багатовіконного серверу X11 на головному процесорі, що виконує прикладне завдання (найбільш відома версія X11 Release 5). X-термінали відрізняються також від стандартних алфавітно-цифрових ASCII і традиційних графічних дисплейних терміналів тим, що вони можуть бути підключені до будь-якої головної системи, яка підтримує стандарт X-Windows. Більш того, локальна обчислювальна потужність X-терміналу зазвичай використовується для обробки відображення, а не обробки прикладних задач (званих клієнтами), які виконуються видалено на головному комп'ютері (сервері). Виведення такого видаленого застосування просто відображується на екрані X-терміналу.

Кластерна архітектура - група об'єднаних між собою обчислювальних машин, що є єдиним вузлом обробки інформації для критично важливих застосувань (обробка транзакцій,

управління базами даних і обслуговування телекомунікацій) з високою продуктивністю і необхідністю тривалого функціонування систем.

Основні характеристики:

- розділення ресурсів: комп'ютери в кластері можуть звертатися до файлів даних як до локальних;
- висока готовність: забезпечується спеціальними контролерами і в разі відмови одного з комп'ютерів або периферійного пристрою його завдання можуть бути перенаправлені на інший комп'ютер або інший пристрій;
- висока пропускна спроможність: ряд прикладних систем можуть користуватися можливістю паралельного виконання завдань на декількох комп'ютерах кластера;
- зручність обслуговування: прикладні програми встановлюються лише один раз на ресурсах, доступних всім комп'ютерам; загальні бази даних можуть обслуговуватися з одного місця;
- розширюваність: підвищення продуктивності досягається простим підключенням додаткових комп'ютерів.

До теперішнього часу усе більш чітко є видимими глобальні зміни у функціональній класифікації комп'ютерів, в результаті яких в цій схемі, можливо, залишаться лише два класи: сервери і клієнти. Усередині цих двох класів, швидше за все, з'являться додаткові градації, наприклад: домашній сервер, сервер локальної мережі, інтернет-сервер і таке інше.

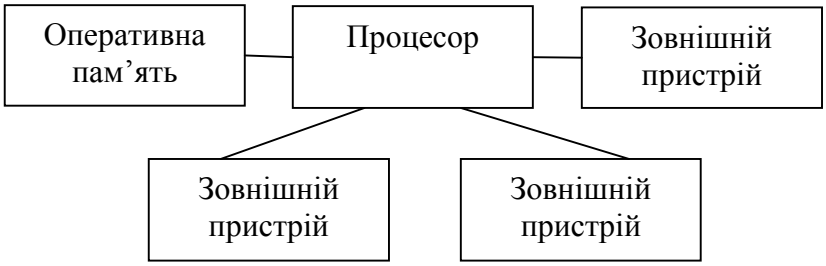
1.3.5. Класифікація комп'ютерів за архітектурою.

Архітектура «зірка».

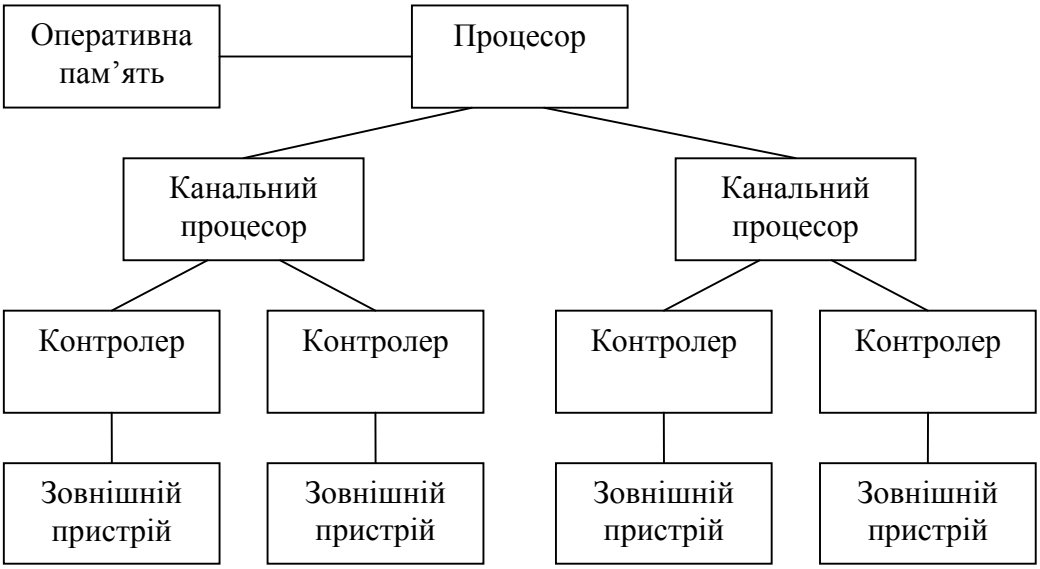
Тут процесор (рис.1.5*а*) поєднується безпосередньо з пристроями введення-виведення та управляє їх роботою (ранні моделі машин). Цей тип також називається класичною архітектурою фон Неймана - один арифметико-логічний пристрій (АЛП), через який проходить потік даних, і один керуючий пристрій (КП), через який проходить потік команд - програма. Це однопроцесорний комп'ютер.

Принстонская і гарвардська архітектури.

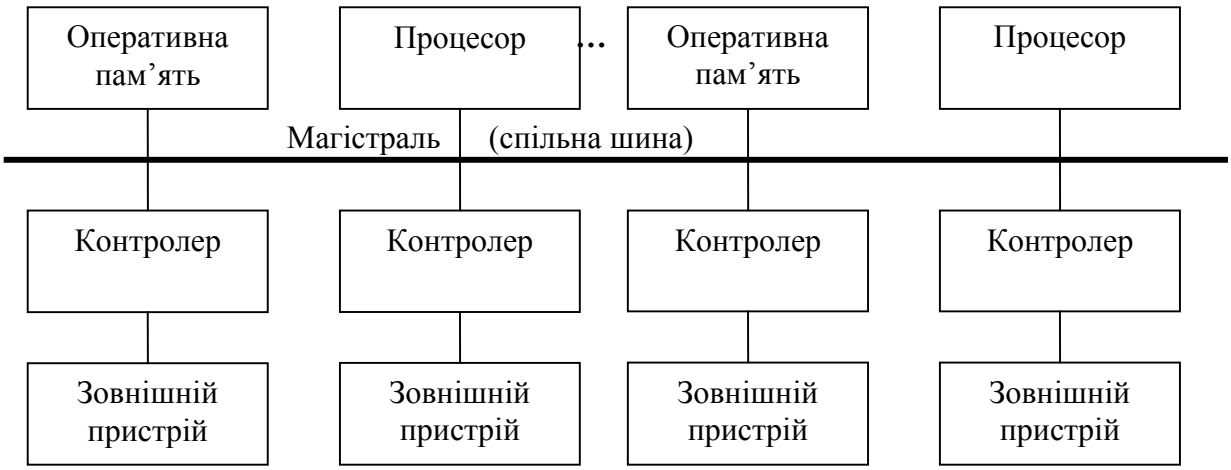
Архітектура фон Неймана часто асоціюється з принстонской архітектурою, яка характеризується використанням загальної оперативної пам'яті для зберігання програм і даних.



а) архітектура «зірка»



б) ієрархічна;



в) магістральна

Рис. 1.5. Основні класи архітектури КС.

Її альтернативою є гарвардська архітектура, яка характеризується фізичним розділенням

пам'яті команд (програм) і пам'яті даних. Кожна пам'ять з'єднується з процесором окремою шиною, що дозволяє одночасно з читанням-записом даних при виконанні поточної команди проводити вибірку та декодування наступної команди.

Гарвардська архітектура використовується в сучасних процесорах, коли в кеш-пам'яті ЦП виділяється пам'ять команд (I-Cache) і пам'ять даних (D-Cache).

Ієрархічна архітектура (рис.1.5б) — ЦП з'єднується з периферійними процесорами (або допоміжними процесорами, каналами, канальними процесорами), які в свою чергу керують контролерами, до яких підключені групи пристроїв введення-виведення (системи IBM 360-375, ЄС ЕОМ).

Магістральна структура (загальна шина — unibas, рис.1.5в). Процесор (або процесори) і блоки пам'яті (ОП) взаємодіють між собою та з периферією через внутрішній канал, загальний для всіх пристроїв (машини DEC, IBM PC-сумісні ПЕВМ). Фізично магістраль є багатопровідною лінією з роз'ємами для підключення електронних схем. Сукупність ліній магістралі розділяється на окремі групи — шину адреси, шину даних і шину управління.

1.4. Шляхи підвищення продуктивності і надійності обчислювальних систем.

1.4.1. Підвищення продуктивності.

Загальний метод підвищення продуктивності полягає в організації паралельних обчислень, тобто одночасне вирішення завдань або поєднанні в часі етапів рішення однієї задачі.

Розрізняють 3 способи організації паралельних обчислень:

1) Поєднання в часі різних етапів різних завдань – можливий в однопроцесорних системах.

2) Одночасне вирішення різних завдань або частин завдання – можливий лише за наявності декількох оброблювальних пристроїв для кожної підзадачі. Тут виділяють декілька типів паралелізму:

Природний паралелізм незалежних завдань: у систему поступає декілька непов'язаних потоків від різних завдань.

Паралелізм незалежних гілок: при рішенні завдання можуть бути виділені окремі незалежні частини (гілки), які за наявності декількох оброблювальних пристроїв можуть виконуватися паралельно і незалежно один від одного. Гілки називаються незалежними, якщо:

- жодна вхідна змінна гілки не є вихідною для іншої гілки (відсутність функціональних зв'язків);
- гілки не виробляють запис в ті самі елементи пам'яті;
- умови виконання однієї гілки не залежить від результатів або ознак, отриманих в іншій гілці (незалежність за керуванням);
- обидві гілки повинні виконуватися по різних блоках програми (програмна незалежність).

Недоліки: складність виділення незалежних гілок. Вирішувані завдання: матрична алгебра, лінійне програмування, спектральна обробка сигналів, прямі і зворотні перетворення Фур'є.

Паралелізм об'єктів або даних: за однією і тією ж програмою повинні оброблятися дані, що отримуються від різних джерел, наприклад, від однотипних датчиків, встановлених на декількох об'єктах.

3) Конвеєрна обробка, при якій операція розбивається на декілька частин, кожна з яких обробляється блоком процесора, що спеціалізується саме на даному типі операцій.

1.4.2. Підвищення надійності.

Важливою характеристикою обчислювальних систем є *надійність* – властивість системи виконувати покладені на неї функції в заданих умовах функціонування.

Важлива характеристика надійності – інтенсивність відмов – середнє число відмов за одиницю часу. Якщо система містить n елементів, а λ_i - інтенсивність відмови i-го блоку, то інтенсивність відмов в системі:

$$\lambda_0 = \sum_{i=1}^{n} \lambda_i .$$

Напрацювання на відмову – середній проміжок часу між двома суміжними відмовами:

$$T_0 = 1 / \lambda_0 .$$

При цьому вірогідність того, що за час t станеться відмова, дорівнює:

$$P(t < x) = 1 - e^{-\frac{t}{T_0}} .$$

Наприклад, якщо $\lambda_0 = 10^{-2}$ годин, то $T_0 = 100$, а $P(t < 100) \approx 0.63$. Тобто з вірогідністю 37% відмова станеться за час більше 100 годин.

Підвищення надійності засноване на принципі запобігання несправностям шляхом зниження інтенсивності відмов та збоїв за рахунок вживання електронних схем і компонентів з високою і надвисокою мірою інтеграції, зниження рівня перешкод, полегшених режимів роботи схем, забезпечення теплових режимів їх роботи, а також за рахунок вдосконалення методів збирання апаратури.

Відмовостійкість - це така властивість обчислювальної системи, яка забезпечує їй як логічній машині можливість продовження дій, заданих програмою, після виникнення несправностей. Введення відмовостійкості вимагає надлишкового апаратного і програмного забезпечення. Напрями, пов'язані із запобіганням несправностям і з відмовостійкістю - основні в проблемі надійності. Концепції паралельності і відмовостійкої обчислювальних систем природним чином зв'язані між собою, оскільки в обох випадках потрібні додаткові функціональні компоненти. Тому, власне, на паралельних обчислювальних системах досягається як найбільш висока продуктивність, так і дуже висока надійність. Наявні ресурси надмірності в паралельних системах можуть гнучко використовуватися як для підвищення продуктивності, так і для підвищення надійності. Структура сучасних багатопроцесорних і многомашинних систем пристосована до автоматичної реконфігурації і забезпечує можливість продовження роботи системи після виникнення несправностей.

Поняття надійності включає не лише апаратні засоби, але і програмне забезпечення. Головною метою підвищення надійності систем є цілісність даних, що зберігаються в них.

Масштабованість є можливістю нарощувати кількість та потужності процесорів, об'ємів оперативної і зовнішньої пам'яті та інших ресурсів обчислювальної системи. Масштабованість повинна забезпечуватися архітектурою і конструкцією комп’ютера, а також відповідними засобами програмного забезпечення.

Додавання кожного нового процесора в дійсно масштабованій системі повинне давати прогнозоване збільшення продуктивності і пропускної спроможності при прийнятних витратах. Одне з основних завдань при побудові масштабованих систем є мінімізація вартості розширення комп'ютера і спрощення планування. У ідеалі додавання процесорів до системи повинне вести до лінійного зростання її продуктивності. Проте це не завжди так. Втрати продуктивності можуть виникати, наприклад, при недостатній пропускній спроможності шин даних через зростання трафіку між процесорами і основною пам'яттю, а також між пам'яттю і

пристроями введення/виводу. Реальне збільшення продуктивності важко оцінити заздалегідь, оскільки воно в значній мірі залежить від динаміки поведінки прикладних задач.

Масштабованість програмного забезпечення зачіпає всі його рівні від простих механізмів передачі повідомлень до роботи з такими складними об'єктами як монітори транзакцій та середовище прикладної системи. Зокрема, програмне забезпечення повинне мінімізувати трафік міжпроцесорного обміну, який може перешкоджати лінійному зростанню продуктивності системи. Апаратні засоби (процесори, шини і пристрої введення/виводу) є лише частиною масштабованої архітектури, на якій програмне забезпечення може забезпечити передбачене зростання продуктивності. Важливо розуміти, що простий перехід, наприклад, на потужніший процесор може привести до перевантаження інших компонентів системи. Це означає, що дійсно масштабована система має бути збалансована по всіх параметрах.

2. Загальні структури високопродуктивних систем.

Термін "архітектура системи" часто вживається як у вузькому, так і в широкому сенсі цього слова. У вузькому сенсі (найчастіше вживаному) під архітектурою розуміється архітектура набору команд. Архітектура набору команд служить кордоном між апаратурою і програмним забезпеченням і представляє ту частину системи, яку видно програмістові або розробникові компіляторів. У широкому сенсі архітектура охоплює поняття організації системи, що включає такі високорівневі аспекти розробки комп'ютера як систему пам'яті, структуру системної шини, організацію введення/виводу і тому подібне.

Стосовно КС термін "архітектура" визначається як розподіл функцій, що реалізовуються системою, між її рівнями, точніше як визначення кордонів між цими рівнями. Таким чином, архітектура обчислювальної системи передбачає багаторівневу організацію. Архітектура першого рівня визначає, які функції по обробці даних виконуються системою в цілому, а які покладаються на зовнішній світ (користувачів, операторів, адміністраторів баз даних і таке інше). Система взаємодіє із зовнішнім світом через набір інтерфейсів: мови (мова оператора, мови програмування, мови опису і маніпулювання базою даних, мова управління завданнями) і системні програми (програми-утиліти, програми редагування, сортування, збереження і відновлення інформації).

Інтерфейси наступних рівнів можуть розмежовувати певні рівні усередині програмного забезпечення. Наприклад, рівень управління логічними ресурсами може включати реалізацію таких функцій, як управління базою даних, файлами, віртуальною пам'яттю, мережевою

телеобробкою. До рівня управління фізичними ресурсами відносяться функції управління зовнішньою і оперативною пам'яттю, управління процесами, що виконуються в системі.

Наступний рівень відображає основну лінію розмежування системи, а саме кордон між системним програмним забезпеченням і апаратурою. Цю ідею можна розвинути і далі і говорити про розподіл функцій між окремими частинами фізичної системи. Наприклад, деякий інтерфейс визначає, які функції реалізують центральні процесори, а які - процесори введення/виведення. Архітектура наступного рівня визначає розмежування функцій між процесорами введення/виведення і контролерами зовнішніх пристроїв. У свою чергу можна розмежувати функції, що реалізовуються контролерами і самими пристроями введення/виведення (терміналами, модемами, накопичувачами на магнітних дисках чи стрічках). Архітектура таких рівнів часто називається архітектурою фізичного введення/виведення.

2.1. Архітектура набору команд.

Під архітектурою системи в вузькому сенсі розуміють архітектуру набору команд. На рис.2.1 зображено сучасну ієрархічну класифікацію типів наборів команд.

Акумуляторна архітектура за фон Нейманом є вихідною точкою розвитку архітектури всіх сучасних цифрових машин. В цій архітектурі основою арифметико-логічного пристрою є єдиний регістр, який називається акумулятором. Типова команда вибирає операнд з оперативної пам'яті і використовує його в операції з акумулятором, де, за необхідністю, вже повинен знаходитися другий операнд. Результат залишається в акумуляторі. Система має дві команди, які можуть працювати з оперативною пам'яттю. Одна команда необхідна для завантаження даних в акумулятор процесору з оперативної пам'яті, а друга — для запису коду з акумулятора в пам'ять.

Типовий приклад послідовності машинних команд для такого процесору має вигляд:

```
load  Pa ;загрузка у акумулятор з ОП
add   Pb ;складання
store Pc ;запис результату в ОП
```

Тут Pa і Pb представляють адреси полів оперативної пам'яті, в яких знаходяться додатки, а Pc — адрес поля для запису результату.

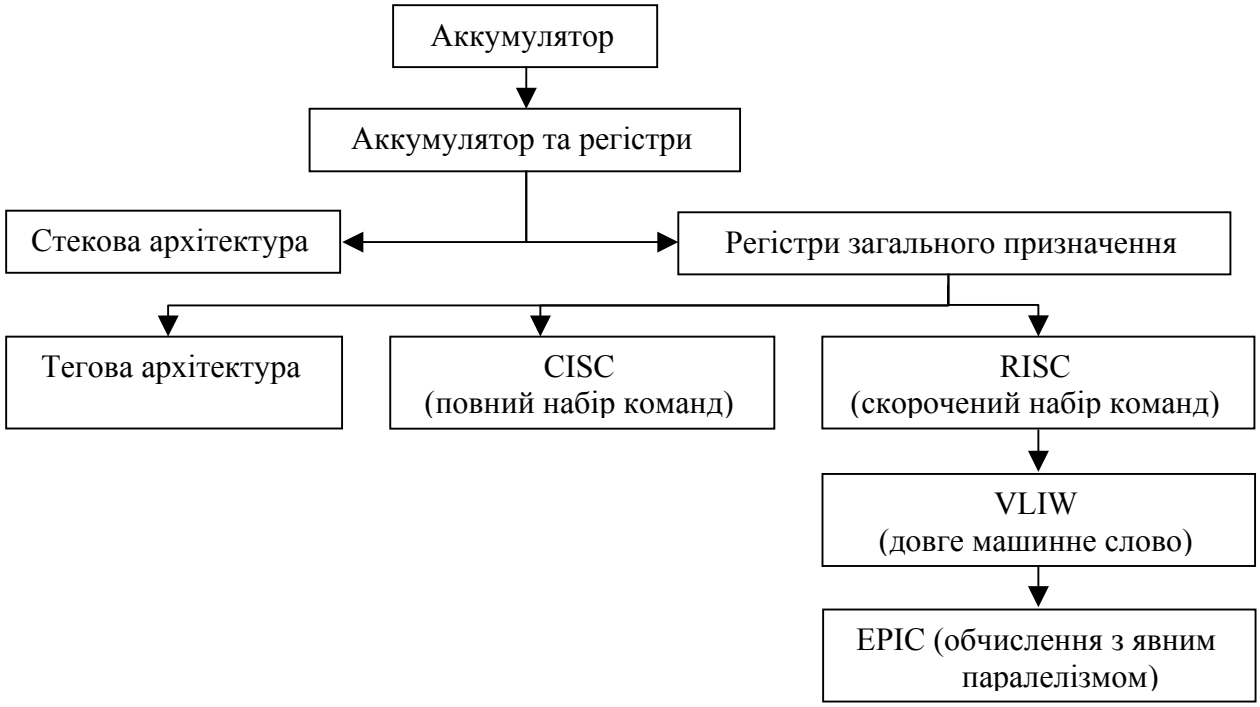


Рис. 2.1. Класифікація систем за архітектурою набору команд.

Подальший розвиток акумуляторної архітектури привів до появи архітектури *акумулятор і регістри*, в якій до складу процесору разом з акумулятором входили декілька додаткових регістрів. Вони використовувалися в основному як індексні.

Команді *стекової* архітектури в основному відносяться до безадресних. Для зберігання операндів використовується стек, що входить до складу процесору. Обидва операнди вибираються з вершини стека, над ними виконується дія, що задається командою, результат залишається як нова вершина стека. В систему команд входять дві одноадресні команди, які забезпечують копіювання даних у вершину стека з поля оперативної пам'яті, а також вибір коду з вершини стека і запис в поле оперативної пам'яті.

```
push Pa ;запис першого операнда у вершину стека з ОП
push Pb ;запис другого операнда у вершину стека з ОП
add     ;складання
pop  Pc ;вибір результату з вершини стека і запис в ОП
```

Стекова архітектура в класичному вигляді не знайшла широкого вживання.

Архітектура *регістрів загального призначення* відрізняється тим, що процесор має в своєму розпорядженні набір високошвидкісних регістрів, який прийнято називати регістровою

пам'яттю або регістровим файлом. На відміну від стекової архітектури, операнди команд можуть вибиратися з будь-якого регістра і записуватися в будь-який регістр процесора. Функціональне, цільове використання регістрів може бути довільним.

Розрізняють дві великі групи архітектури регістрів загального призначення: з повним та скороченим набором команд. Ці групи є ідеалізованими (модельними). Будь-який реальний процесор фактично займає деяке проміжне положення з переважанням особливостей тієї або іншої моделі.

До архітектури регістрів загального призначення відноситься і так звана тегова архітектура, відмітною особливістю якої є наявність пов'язаного з кожним стандартним полем оперативної пам'яті апаратно керованого тегу. У даному випадку тег - це деяка ознака, що визначає типи даних, припустимих для поля, а також множиною команд, що застосовуються до нього. Тегову архітектуру має система команд вітчизняної обчислювальної системи «Ельбрус».

CISC-архітектура. До середини 1990-х рр. переважаючою вважалася архітектура, яку прийнято позначати абревіатурою CISC (Complete Instruction Set Computer), тобто комп'ютер з повним набором інструкцій (машинних команд). Особливістю цього різновиду архітектури є наявність окремої машинної команди для кожної можливої дії з обробки даних.

Для архітектури CISC характерно:

- порівняно невелике число регістрів загального призначення;

- велика кількість машинних команд, деякі з яких навантажені семантично аналогічно операторам високорівневих мов програмування та для виконання яких необхідно декілька тактів роботи процесора;

- велика кількість методів адресації;

- велика кількість форматів команд різної розрядності;

- переважання двоадресного формату команд;

- наявність команд обробки типа регістр-пам'ять;

- розвинений механізм адресації операндів, що включає різні методи непрямої адресації.

До архітектури CISC відносяться системи команд сімейств IBM 360/370, VAX, Intel 80x86, в тому числі система команд машини IBM PC з процесором i8086. Слід зазначити, що в останніх розробках компанії Intel широко використовуються ідеї, реалізовані в risc-мікропроцесорах, так що багато відмінностей між CISC і RISC нівелюються. Проте складність архітектури і системи команд x86 залишається і є головним чинником, що обмежує продуктивність процесорів на її основі.

RISC-архітектура (від Reduced Instruction Set Computer або Code) - комп'ютер із скороченим набором інструкцій. В цьому підході передбачається включення в систему команд процесора лише простих дій, що часто зустрічаються. Реалізація складніших операцій над даними здійснюється за допомогою послідовностей простих команд.

Система команд такого комп'ютера містить лише мікрокоманди. Для архітектури RISC характерно:

- вона побудована на архітектурі, що відокремлює команди обробки від команд роботи з пам'яттю;
- упор на ефективну конвеєрну обробку;
- система команд розроблялася так, щоб виконання будь-якої команди займало невелику кількість машинних тактів (переважно один машинний такт);
- з метою підвищення продуктивності логіка виконання команд орієнтується на апаратну, а не на мікропрограмну реалізацію;
- для спрощення логіки декодування команд використовуються команди фіксованої довжини і фіксованого формату;
- великий регістровий файл: 32 і більше регістрів загального призначення;
- часте використання триадресних команд, що додатково дає можливість зберігати велику кількість змінних в регістрах без їх подальшого перезавантаження;
- використання лише простих способів адресації (регістрова, пряма, безпосередня).

Для комп'ютерів, що відносяться до RISC-архітектури, потрібні досконаліші компілятори. До RISC-архітектури відносять сімейства SUN SPARC, Alpha, Power PC, MIPS і деякі інші.

Архітектура VLIW (від Very Large Instruction Word - дуже довге командне слово) є різновидом RISC- архітектури. Архітектура процесорів цієї групи - суперскалярна, з наявністю великої кількості арифметико-логічних функціональних блоків. Основною її відмінністю є можливість об'єднання декількох простих команд в так звану зв'язку. Команди, що до неї входять, мають бути незалежні одна від одного, тобто їх можна виконувати одночасно, паралельно. Таким чином, з декількох незалежних машинних команд транслятор формує одне «дуже довге командне слово». Наприклад, в системі команд вітчизняного суперкомп'ютера «Ельбрус-3» командне слово займає 256 біт в упакованому вигляді та 500 біт у розпакованому. Одне командне слово задає до семи арифметичних або логічних операцій.

Підкреслімо, що аналіз можливості об'єднання машинних команд у зв'язку здійснюється на стадії трансляції. Тому під час виконання програми процесору не потрібно виконувати переупорядковування команд з метою виключення простоїв конвеєра. Процесорі архітектури

Табл.2.1. Типи адресації.

Метод адресації	Приклад команди	Сутність методу команди	Використання
Регістрова	Add R4,R3	R4<-R4+R3	Необхідні значення знаходяться в регістрах;
Безпосередня або літеральна	Add R4,#3	R4<-R4+3	Для завдання констант;
Базова із зсувом	Add R4,100(R1)	R4<-R4+Mem[R1+100]	Для звернення до локальних змінних;
Непряма регістрова адресація	Add R4,(R1)	R4<-R4+Mem[R1]	Для звернення по покажчику або обчисленій адресі;
Індексна	Add R3,(R1+R2)	R3<-R3+Mem[R1+R2]	Корисна при роботі з масивами: R1 – база, R2 – індекс;
Пряма або абсолютна	Add R3,(1000)	R3<-R3+Mem[1000]	Звернення до статичних даних;
Непряма	Add R1,@(R3)	R1<-1+Mem[Mem[R3]]	R3 – адрес покажчика p, за яким обирається значення;
Автоінкрементна	Add R1,(R2)+d	R1<-R1+Mem[R2]; R2<-R2+d	Робота в циклі з масивами: R2 – початок масиву, d – зсув індексу на кожному циклі;
Автодекрементна	Add R1,(R2)-d	R1<-R1+Mem[R2]: R2<-R2-d	Робота в циклі з масивами: R2 – початок масиву, d – зміщення індексу на кожному циклі;
Базова індексна із зсувом і	Add R1,(100)R2[R3]	R1<-R1+Mem[100]+	Індексація масивів.

VLIW використовують спеціальну систему команд і тому несумісна з системою команд процесорів сімейства Intel 80x86.

Архітектура EPIC (від Explicitly Parallel Instruction Computing - обробка команд з явним паралелізмом) є розвитком VLIW-архітектури. Вона була розроблена Intel і Hewlett-Paccard (HP). Відмінною рисою EPIC- архітектури є усунення відмічених недоліків VLIW-архітектури, що вимагали, наприклад включення групи порожніх команд для заповнення машинних тактів, що виникають при реалізації паралельного виконання деяких команд.

Характерні особливості EPIC- архітектури:

- гарна масштабованість функціональних блоків процесора;
- паралелізм, що явно задається в машинному коді;

- предикативне виконання команд.

Кількість процесорів, що входять до складу однієї обчислювальної системи, вже в даний час обчислюється десятками тисяч. Тому проблемі масштабованості в архітектурі EPIC приділяється багато уваги.

Явне завдання паралелізму в команді EPIC-архітектури покликане спростити і прискорити роботу процесора, а предикативне виконання фактично перетворює обробку розгалужень на паралельне виконання декількох лінійних фрагментів програми, що позбавляє конвеєрну схему від проблем з очищенням та заповненням конвеєру.

2.2. Методи адресації.

У машинах з регістрами загального призначення метод (або режим) адресації об'єктів, з якими маніпулює команда, може задавати константу, регістр або елемент пам'яті. Для звернення до комірки пам'яті процесор, перш за все, повинен обчислити дійсну або ефективну адресу пам'яті, яка визначається заданим в команді методом адресації.

У табл.2.1. наведено основні типи адресації.

2.3. Типи команд.

Команди традиційного машинного рівня можна розділити на кілька типів, які наведено в табл.2.2.

Табл.2.2. Типи машинних команд.

Тип операції	Приклади
Арифметичні та логічні	Цілочисельні арифметичні і логічні операції: додавання, віднімання, логічне додавання, логічне множення і т.і.;
Пересилання даних	Операції завантаження / запису;
Керевання потоком команд	Безумовні і умовні переходи, виклики процедур та повернення;
Системні операції	Системні виклики, команди управління віртуальною пам'яттю і т.і.;
Операції з плаваючою точкою	Операції складання, віднімання, множення і ділення дійсних чисел;
Десяткові операції	Десяткове додавання, множення, перетворення форматів і т.і.;
Операції над рядками	Пересилання, порівняння і пошук рядків.

3. Особливості організації пам'яті, процесорів, інтерфейсів і підсистем введення-виведення.

3.1. Пам'ять

3.1.1. Загальні відомості про пам'ять.

Пам'ять ЕОМ - сукупність пристроїв, які необхідні для запам'ятовування, зберігання і видачі інформації. Окремі пристрої з даної сукупності називаються запам'ятовуючими пристроями (ЗП).

Найважливіші характеристики пам'яті:

- ємність пам'яті - максимальне число даних, які можуть зберігатися;
- питома ємкість пам'яті - відношення ємності ЗП до його фізичного об'єму;
- швидкодія пам'яті - тривалість операцій звернення: пошуку інформації та зчитування або пошуку адреси і запису.

За способом організації розрізняють:

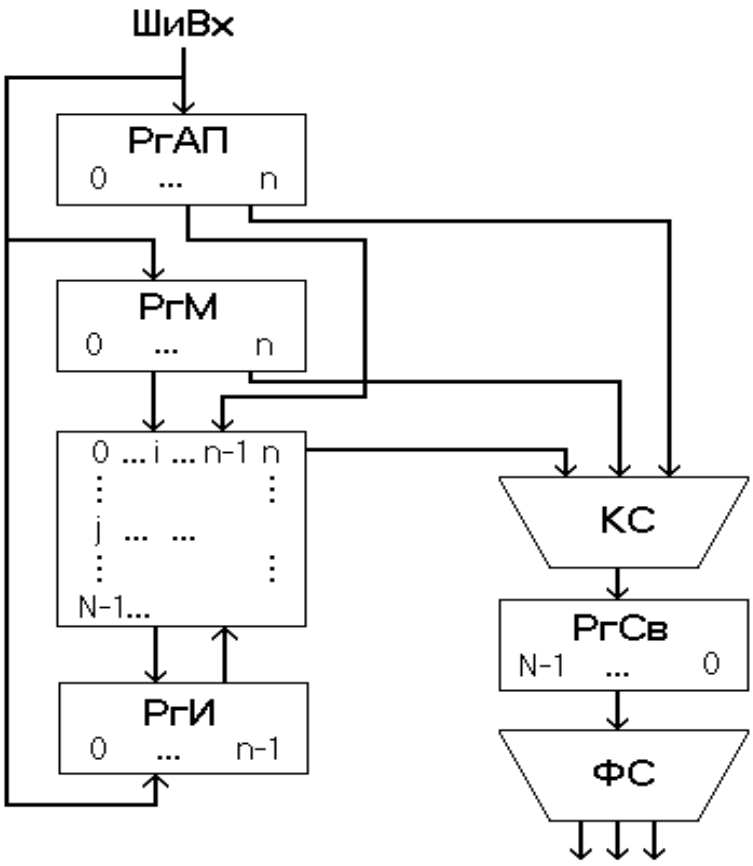
- пам'ять з довільним доступом, при якому цикл звернення не залежить від місця розташування комірки пам'яті, до якої проводиться доступ. Число розрядів, що прочитуються за один прохід, називається шириною вибірки;
- пам'ять з послідовним доступом, при якому проводиться послідовне переглядання ділянок носія інформації, поки необхідна комірка не займе вихідне положення. Приклади: дискові накопичувачі, накопичувачі на магнітній стрічці.

Спосіб організації пам'яті залежить від методів розміщення і пошуку інформації.

Адресна пам'ять: пошук і розміщення інформації засновані на використанні адреси слова, що зберігається. Адресою служить номер елементу пам'яті.

Асоціативна пам'ять: пошук потрібної інформації проводитися не за адресою, а за її вмістом (рис.3.1). При цьому пошук за ознакою відбувається паралельно для всіх комірок запам'ятовуючого пристрою. В ряді випадків це дозволяє істотно спростити пошук інформації за рахунок поєднання операцій зчитування з низкою логічних операцій. Наприклад, в пам'яті зберігаються дані про успішність студентів. За один запит до даної пам'яті можна визначити, наприклад, скільки студентів мають відмінні оцінки.

Стекова пам'ять: сукупність комірок, що створюють одновимірний масив, в якому



ШиВх - вхідна інформаційна шина;
РгАП - регістр асоціативного пошуку;
РГМ - рагістр маски;
КС - комбінаційна схема;
РгСв - регістр збігів;
Пошук здійснюється за розрядами з РгАП, для яких біт РГМ встановлений в1. Для тих, що збігаються КС встановлює в 1 розряд в РгСВ. ФС формує з РгСв сигнали, відповідні випадкам збігу.

Рис. 3.1. Робота асоціативної пам'яті.

сусідні комірки зв'язані і можуть передавати інформацію одна інший. Запис нового слова проводитися у верхню комірку. При цьому всі дані зрушуються на одну комірку вниз. Зчитування проводитися в зворотному порядку.

3.1.2. Ієрархія пам'яті. Організація кеш-пам'яті

Ієрархія пам'яті сучасних комп'ютерів будується на декількох рівнях, причому вищий рівень менше за об'ємом, швидше і має більшу вартість в перерахунку на байт, ніж нижчий

рівень. Рівні ієрархії взаємозв'язані: всі дані на одному рівні можуть бути також знайдені на нижчому рівні і всі дані на цьому нижчому рівні можуть бути знайдені на наступному рівні, що є нижчим і так далі, поки мі не досягнемо основи ієрархії.

Ієрархія пам'яті зазвичай складається з багатьох рівнів, але в кожен момент ми маємо справу лише з двома сусідніми рівнями. Мінімальна одиниця інформації, яка може бути або присутньою, або бути відсутньою у дворівневій ієрархії, називається блоком. Розмір блоку може бути або фіксованим, або змінним. Якщо цей розмір зафіксований, то об'єм пам'яті є кратним розміру блоку.

Успішне або неуспішне звернення до вищого рівня називаються відповідно *попаданням* (hit) або *промахом* (miss). Попадання - є звернення до об'єкту в пам'яті, який знайдений на більш високому рівні, тоді як промах означає, що він не знайдений на цьому рівні. Доля попадань (hit rate) або коефіцієнт попадань (hit ratio) це доля звернень, знайдених на більш високому рівні. Інколи вона представляється відсотками. Доля промахів (miss rate) це доля звернень, які не знайдені на більш високому рівні.

Оскільки підвищення продуктивності є головною причиною появи ієрархії пам'яті, частота попадань і промахів є важливою характеристикою.

Час звернення при попаданні – час звернення до більш високого рівня ієрархії, який додатково містить час, який необхідно для перевірки того, чи є звернення попаданням чи промахом.

Втрата при промаху – час на заміщення блоку на вищому рівні на блок з нижчого рівня та час на пересилання цього блоку в необхідний пристрій.

3.1.3. Організація кеш-пам'яті

Ефективність тієї чи іншої системи побудови кеш-пам’яті залежить від стратегії керування нею. Цю стратегію можна конструктивно задати, давши відповідь на наступні чотири питання.

1. Де може розміщуватися блок в кеш-пам'яті?

Принципи розміщення блоків в кеш-пам'яті визначають три основні типи їх організації:

Якщо кожен блок основної пам'яті має лише одне фіксоване місце, на якому він може з'явитися в кеш-пам'яті, то така кеш-пам'ять називається *кешем з прямим відображенням* (direct mapped). Це найбільш проста організація кеш-пам'яті, при якій для відображення адресу блоків основної пам'яті на адреси кеш-пам'яті просто використовуються молодші розряди

адреси блоку. Таким чином, всі блоки основної пам'яті, що мають однакові молодші розряди в своїй адресі, потрапляють в один блок кеш-пам'яті, тобто

(адреса блоку кеш-пам'яті) =

(адреса блоку основної пам'яті) mod (число блоків в кеш-пам'яті).

Такий спосіб організації є найбільш простим. Головним його недоліком є те, що за єдине місце в кеш-пам’яті можуть сперечатися декілька областей з основної пам’яті.

Якщо деякий блок основної пам'яті може розташовуватися на будь-якому місці кеш-пам'яті, то кеш називається *повністю асоціативним* (folly associative).

Якщо деякий блок основної пам'яті може розташовуватися на обмеженій множині місць в кеш-пам'яті, то кеш називається *множинно-асоціативним* (set associative). Зазвичай множина є групою з двох або більшого числа блоків в кеші. Якщо множина складається з *n* блоків, то таке розміщення називається множинно-асоціативним з *n* каналами (*n*-way set associative). Для розміщення блоку перш за все необхідно визначити множину за молодшими розрядами адреси блоку пам'яті (індексом):

(адреса множини кеш-пам'яті) =

(адреса блоку основної пам'яті) mod (кількість множин в кеш-пам'яті)

Далі, блок може розміщуватися на будь-якому місці даної множини.

2. Як знайти блок, що знаходиться в кеш-пам'яті?

У кожного блоку в кеш-пам'яті є адресний тег, що вказує на блок в основній пам'яті. Ці теги зазвичай одночасно порівнюються з виробленою процесором адресою блоку пам'яті.

Крім того, необхідний спосіб визначення того, що блок кеш-пам'яті містить достовірну або придатну для використання інформацію. Найбільш загальним способом вирішення цієї проблеми є додавання до тегу так званого біта достовірності (valid bit).

3. Який блок кеш-пам'яті має бути заміщений при промаху?

При виникненні промаху, контролер кеш-пам'яті повинний вибрати блок, що підлягає заміщенню. При використанні організації кешу з прямим відображенням на попадання перевіряється лише один блок і лише цей блок може бути заміщений. При повністю асоціативній або множинно-асоціативній організації кеш-пам'яті є декілька блоків, з яких треба вибрати кандидата в разі промаху. Як правило, для заміщення блоків застосовуються дві основні стратегії: випадкова і LRU.

Випадковий вибір: блоки-кандидати обираються випадково, аби мати рівномірний розподіл. В деяких системах використовують псевдовипадковий алгоритм заміщення.

Використання останнього: аби зменшити вірогідність викидання інформації, яка скоро може бути потрібною, всі звернення до блоків фіксуються. Замінюється той блок, який не використовувався довшим за всіх (LRU - Least-Recently Used).

Перевага випадкового способу полягає в тому, що його простіше реалізувати в апаратурі. Колі кількість блоків збільшується, алгоритм LRU стає все більш дорогим і лише наближеним.

4. Що відбувається під час запису?

При зверненнях до кеш-пам'яті на реальних програмах переважають звернення по читанню. Всі звернення за командами є зверненнями по читанню і більшість команд не пишуть в пам'ять. Зазвичай операції запису складають менше 10% загального трафіку пам'яті. Тому бажано оптимізувати кеш-пам'ять для виконання операцій читання, проте при реалізації високопродуктивної обробки даних не можна нехтувати і швидкістю операцій запису.

Оптимізація при читанні є більш прозорою. Блок з кеш-пам’яті читається у той самий час, коли й його тег. Якщо при цьому читання виконано з попаданням, то блок відразу передається в процесор. Якщо читання виконано з промахом, то від прочитаного блоку немає ніякої користі.

Операції запису тривають, зазвичай, довше. Тут принципово можливі два підходи:

1) *наскрізний запис* (write through, store through) - інформація записується зразу в два місця: в кеш-пам'ять та в основну пам'ять;

2) *запис із зворотним копіюванням* (write back, copy back, store in) – інформація записується тільки в кеш-пам'ять. Модифікований блок кеш-пам’яті записується до основної пам’яті лише в той час, коли він заміщується.

Обидва підходи мають свої переваги та недоліки.

Переваги наскрізного запису:

- промахи по читанню не впливають на запис в більш високий рівень;
- наскрізний запис є більш простим для апаратної реалізації, ніж запис із зворотним копіюванням;
- основна пам'ять містить найбільш свіжу копію даних, що забезпечує когерентність пам’яті в мультипроцесорних системах, а також важливо для введення-виведення.

Переваги запису зі зворотним копіюванням:

- операції запису виконуються зі швидкістю кеш-пам’яті;
- декілька операцій запису в ту саму комірку кеш-пам’яті потребують лише однієї операції запису в основну пам'ять;
- оскільки в цьому випадку звернення до основної пам’яті виконуються рідше, то необхідна менша смуга пропускання пам’яті, що веде по переваг у багатопроцесорних системах.

Запис із зворотним копіюванням має три алгоритмічні модифікації.

В першому *алгоритмі простого свопінгу* при промаху виконується дві операції доступу до основної пам’яті: спочатку необхідно зберегти блок з кеш-пам’яті, який буде замінено, а потім переслати в кеш необхідний блок.

В другому *алгоритмі свопінгу з флагами* додатково відстежується, чи є блок в кеш-пам’яті актуальним, тобто до нього був доступ лише за читанням. Якщо так, то немає необхідності при промаху пересилати цей блок в основну пам'ять, що зменшує кількість звернень до основної пам’яті. Якщо в цей блок в кеші був запис даних, то пересилання все ж таки є необхідним. Для відстеження цього факту додають *флаг-біт*.

В третьому *алгоритмі регістрового стопінгу з флагами* додаються регістри тимчасового зберігання блоків між кеш та основною пам’яттю. В цьому випадку, коли доступ до кешу пройшов з промахом, блок з кеш-пам’яті пересилається в регістр, блок з основної пам’яті переміщується в кеш та надсилається на обробку в процесор; блок з регістру тимчасового зберігання надсилається в основну пам'ять, коли з’явиться така можливість. Така побудова взаємодії між кешем та основною пам’яттю додатково підвищує продуктивність.

В багатопроцесорних системах, в яких кожен процесор має свій кеш, виникає проблема когерентності кеш-пам’яті. Найбільш вживаними протоколами підтримки когерентності кеш-пам’яті є наступні.

MESI (Modified, Exclusive, Shared, Invalid). Процесор розрізняє блоки, яких немає в кешах інших процесорів (Exclusive), та такі, що присутні більш ніж в одному кеші (Shared). Якщо змінюється Shared-блок, то іншим процесорам надсилається сигнал зробити проти своїх записів цього блоку примітку “невірний” (Invalid). При зміні Exclusive-блоку в цьому немає необхідності. Блок отримує примітку “модифікований” (Modified), та заміщується у звичайному порядку.

MOESI (Modified, Owner, Exclusive, Shared, Invalid). Новий атрибут Owner відповідає модифікованому блоку, який містить дані, що не були записані в основну пам'ять та які присутні в кешах інших процесорів. В той час, коли в протоколі MESI процесори не використовують кеші інших процесорів, в протоколі MOESI довільна операція зчитування супроводжується перевіркою кешей інших процесорів. Якщо необхідні дані знаходяться в одному з таких кешей, то вони зчитуються прямо звідти. При цьому запис цих даних в основну пам'ять не відбувається, а блок отримує атрибут Owner.

3.1.3.1. Підвищення продуктивності кеш-пам’яті.

Формула середнього часу доступу до пам’яті із кешем має вигляд:

*Середній час доступу = час звернення при попаданні + доля промахів*втрати при промаху.*

Ця проста формула демонструє шляхи оптимізації роботи кеш-пам’яті: зменшення долі промахів, зменшення втрат при промахах та зменшення часу звернення при попаданнях.

В табл.3.1 наведено основні методи, що використовуються на практиці для підвищення продуктивності кеш-пам’яті.

Табл.3.1. Методі підвищення продуктивності кеш-пам'яті.

Метод	Складність апаратури	Примітки
збільшення розміру блоку	0	
підвищення ступеня асоціативності	1	
кеш-пам'ять з допоміжним кешем	2	
псевдоасоціативні кеші	2	
апаратна попередня вибірка команд і даних	2	попередня вибірка ускладнена
попередня вибірка під управлінням компілятора	3	вимагає неблокованої кеш-пам'яті
спеціальні методи зменшення промахів	0	питання ПО
установка пріоритетів промахів по читанню над записуванням	1	просто для однопроцесорних систем
кеші другого рівня	2	дороге обладнання
прості кеші малого розміру	0	
конвеєризація операцій запису для швидкого попадання при записі	1	

3.1.4. Принципі організації основної пам'яті в сучасних комп'ютерах.

3.1.4.1. Загальні положення.

Основна пам'ять є наступним рівнем ієрархії пам'яті. Вона задовольняє запити кеш-пам'яті і служить як інтерфейс введення/виведення, оскільки є місцем призначення для введення і джерелом для виведення. Для оцінки продуктивності основної пам'яті використовуються два основні параметри: затримка і полоса пропускання.

Затримка пам'яті оцінюється двома параметрами: часом доступу (access time) і тривалістю циклу пам'яті (cycle time).

Час доступу - проміжок часу між видачею запиту на читання і моментом отримання запитаного слова з пам'яті.

Тривалість циклу пам'яті - мінімальний час між двома послідовними зверненнями до пам'яті.

Основна пам'ять сучасних комп'ютерів реалізується на мікросхемах статичних та динамічних ЗПДВ (запам’ятовуючий пристрій з довільною вибіркою).

В основній пам'яті практично будь-якого комп'ютера, проданого після 1975 року, використовувалися напівпровідникові мікросхеми динамічні ЗПДВ (ДЗПДВ), а для побудови кеш-пам'яті при цьому застосовувалися статичні ЗПДВ (СДЗПДВ).

Коротке порівняння мікросхем ДЗУПВ і СЗУПВ наведено в табл.3.2.

3.1.4.2. Методи збільшення продуктивності пам'яті.

Табл.3.2.Порівняння мікросхем ДЗПДВ і СЗПДВ.

	Переваги	**Недоліки**
ДЗПДВ	велика ємкість (в 4-8 разів); менша вартість; всі біти в рядку можна регенерувати одночасно;	необхідна регенерація (до 5% часу); функції регенерації вимагають додаткової апаратури; під час регенерації пам'ять недоступна;
СЗПДВ	менший час доступу; час доступу збігається з тривалістю циклу; немає необхідності в регенерації;	велика вартість; дорожчі (на порядок);

1) *Збільшення розрядності основної пам'яті.*

Недоліком є необхідність узгодження основної пам'яті і процесора. Також стримуючим фактором є обмежена швидкодія шин пам'яті.

2) *Пам'ять з розшаруванням.*

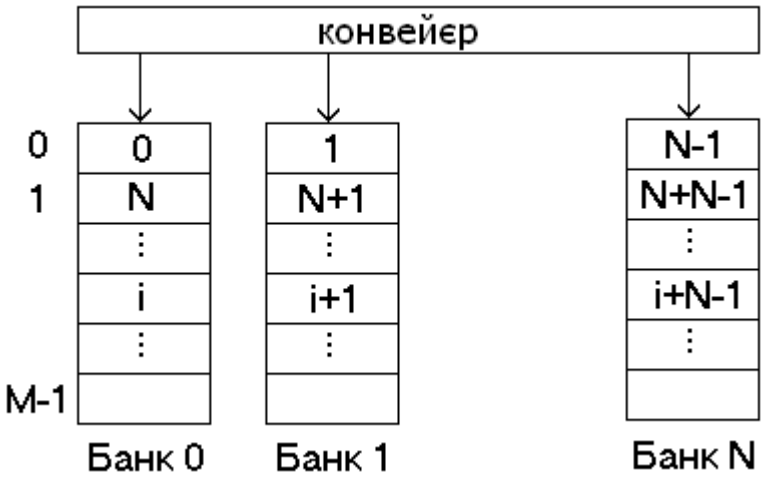


Рис. 3.2 Розшарування пам'яті по банках

Мікросхеми пам'яті часто об'єднуються в банки або модулі, що містять фіксоване число слів, причому, лише до одного з цих слів банку можливе звернення в кожен момент часу.

Отже, аби отримати велику швидкість доступу, потрібно здійснювати одночасний доступ до багатьох банків пам'яті (рис.3.2). Одна із загальних методик, що використовується для цього, називається розшаруванням пам'яті. При розшаруванні банки пам'яті зазвичай упорядковуються так, щоб *N* послідовних адрес пам'яті *i, i+1, i+2 ..., i+N-1* доводилися на *N* різних банків. В *i-му* банку пам'яті знаходяться лише слова, адреси яких мають вигляд k*N + i (де *0<k<M-1*, а *M* число слів в одному банку). Можна досягти в N разів більшої швидкості доступу до пам'яті в цілому, чим до окремого її банку, якщо забезпечити при доступі звернення до даних в різних банках.

3) Використання специфічних властивостей динамічних пристроїв, що запам'ятовують.

Для того, щоб з’ясувати механізми поліпшення роботи динамічної пам’яті, спочатку розглянемо основний цикл доступу:

1. На адресні лінії мікросхеми пам'яті подається адреса рядка. Разом з цим подається сигнал RAS#, який поміщає адрес в буфер адреси рядка.

2. Після стабілізації сигналу RAS#, декодер адреси рядка вибирає потрібний рядок і його вміст переміщається в підсилювач рівня (при цьому логічний стан рядка масиву інвертується).

3. На адресні лінії мікросхеми пам'яті подається адрес стовпця разом з подачею сигналу CAS#, що поміщає адрес в буфер адреси стовпця.

4. Оскільки сигнал CAS# також є сигналом виведення даних, в міру його стабілізації підсилювач рівня відправляє вибрані (відповідно адресу стовпця) дані в буфер виводу.

5. Сигнали CAS# і RAS# послідовно дезактивуються, що дозволяє відновити цикл доступу (після проміжку часу, протягом якого дані з підсилювача рівня повертаються назад до масиву комірок рядка, відновлюючи його колишній логічний стан).

При використанні оптимізації використовують різні види оптимізації, що характерні для різних типів пам’яті.

Мікросхеми FPM DRAM *(Fast Page RAM) – динамічна пам'ять зі швидким сторінковим режимом, розроблена у 1982 році.*

Якщо зафіксувати деякий рядок мікросхеми (або банка), а потім вибирати комірки лише з цього рядка, то можна заощадити час за рахунок відмови від формування адреси рядка, одного і того ж у всіх оброблюваних комірок. Це реалізується наступним чином: після установки адреси рядка цей сигнал не змінюється для подальших декількох змін CAS#-стовбця. Вибраний рядок називається відкритою сторінкою.

Якщо для стандартної пам’яті цикл доступу до чотирьох послідовних комірок має вигляд 5-5-5-5, показуючи необхідність виконання п’яти вищезгаданих етапів доступу до пам’яті, то для FP DRAM цей цикл має вигляд 5-3-3-3. При доступі до одного рядка час складає 14 тактів, що суттєво менше 20 тактів для стандартної DRAM.

Особливість полягає в тому, що виграш досягається лише в тому випадку, якщо програмі дійсно потрібні дані з чотирьох послідовних значень. Якщо доступ здійснювати у випадковому порядку, то виграш не відбувається. Але часто необхідні дані з послідовних комірок пам’яті, наприклад при обробці масивів, тому FP DRAM в цілому ефективніша за DRAM.

Мікросхеми EDO DRAM (Extended Data Out DRAM) – динамічна пам'ять із розширеним затриманням даних на виході, розроблена у 1995 році.

Поліпшення у роботі обумовлене поєднанням в часі зчитування даних з виходу мікросхеми і фази формування адреси наступного стовпця в одному і тому ж циклі пам'яті. Цикл пам'яті цього режиму має структуру 5-2-2-2, і, таким чином, чотири послідовні зчитування вимагають всього 11 тактів, що дає виграш близько 15% у порівнянні з FP DRAM. Якщо доступ проводиться у випадковому порядку, то така пам'ять не відрізняється від звичайної, та виграшу немає.

Мікросхеми BEDO DRAM (Burst EDO DRAM) – розширений час утримування даних на виході із блоковим доступом.

Наступнім кроком стала поява так званого пакетного або блокового, режиму роботи. Блоковій режим полягає в тому, що під час читання однієї комірки разом з нею відбувається читання ще трьох, розташованих в тому ж рядку, причому незалежно від наявності або

відсутності запиту на них. Така група комірок називається пакетом, або банком. Тобто для кожного сигналу RAS# формується лише один сигнал CAS# для першої комірки блоку. Цикл доступу до пам’яті в цьому випадку має вигляд 5-1-1-1, та для зчитування чотирьох послідовних комірок пам’яті потрібно лише 8 тактів.

Мікросхеми SDRAM (Synchronous DRAM) – синхронні DRAM, запропоновані ринку у 1997 році.

В попередніх видах пам'яті моменти видачі сигналів, що управляють і обмін даними у них можуть починатися в довільні моменти часу. Нове поліпшення пов'язане з тим, що в SDRAM всі такти часу строго прив'язані до генератора, при цьому усувається більшість тактів очікування процесора. Орієнтація на тактові імпульси у синхронізованій пам'яті дозволяє організувати поєднання в часі для значно більшої кількості етапів циклу пам'яті, запустивши їх конвеєрне виконання, а також збільшити тривалість поєднань в часі. Окрім синхронного методу читання/запису в мікросхемах SDRAM використовується розшарування пам'яті на незалежні банки, що дозволяє поєднувати вибірку з одного банку з установкою адреси в іншому банку. Крім того, SDRAM підтримує блоковий обмін, характерний для мікросхем BEDO DRAM.

Ефективність мікросхем SDRAM значно вища, ніж EDO або BEDO. Хоча цикл пам'яті має вигляд 5-1-1 -1, тобто чотири операції проходять так само, як і у BEDO, за 8 тактів, SDRAM має тривалість циклу 10 нс, тоді як у BEDO DRAM – 20нс.

Мікросхеми DDR SDRAM (Double Data Rate SDRAM) – синхронізована динамічна пам'ять з подвоєнням даних.

Цей вид пам’яті з’явився внаслідок покращень архітектури SDRAM, тому зустрічається і інше його назва - SDRAM II. Подвоєння швидкості досягається не лише за рахунок збільшення тактової частоти, але і за рахунок передачі даних два рази за один цикл пам'яті: перший раз передача відбувається на початку циклу, а другий - в його кінці. Подальшим розвитком DDR SDRAM є стандарт DDR2 SDRAM, в якому передача даних здійснюється вже чотири рази за цикл. Цикл доступу становить 5-6нс.

3.1.5. Віртуальна пам'ять і організація захисту пам'яті.

У будь-який момент часу комп'ютер виконує безліч процесів або завдань, кожен з яких має свій адресний простір. Тому необхідний механізм поділу невеликої фізичної пам'яті між окремими завданнями. Механізм віртуальної пам'яті ділить фізичну пам'ять на блоки і

розподіляє їх між різними завданнями. При цьому передбачається також деяка схема захисту, яка обмежує завдання тими блоками, які йому належать. Більшість типів віртуальної пам'яті скорочують також час початкового запуску програми на процесорі, оскільки не весь програмний код і дані потрібні їй у фізичній пам'яті, щоб почати виконання.

Якщо програма стає занадто великою для фізичної пам'яті, її частину необхідно зберігати у зовнішній пам'яті (на диску). Механізм віртуальної пам'яті автоматично керує двома рівнями ієрархії пам'яті: основною пам'яттю та зовнішньою (дисковою) пам'яттю.

Системи віртуальної пам'яті можна розділити на два класи: системи з фіксованим розміром блоків, званих *сторінками*, і системи зі змінним розміром блоків, званих *сегментами*.

На даних поняттях також заснована організація захисту пам'яті. Для кожної сторінки (сегмента) є атрибут, який показує якій задачі він належить, і права доступу (запис, читання, запис-читання). Доступ можливий тільки відповідно до атрибутів. Для організації взаємодії задач необхідні спеціальні механізми: організація сторінок пам'яті з особливим доступом, копіювання загальних процедур, тощо.

3.2. Процесори.

3.2.1. Загальна структура процесора.

Процесором називається пристрій, що безпосередньо здійснює процес обробки даних і програмне управління цим процесом. Процесор дешифрує і виконує команди програми, організовує звернення до оперативної пам'яті, в потрібних випадках ініціює роботу зовнішніх пристроїв, сприймає і обробляє запити, що поступають з пристроїв машини та із зовнішнього середовища. Він здійснює управління взаємодією всіх пристроїв, що входять до складу ЕОМ.

Виконання команд, зазвичай, ділиться на дрібніші команди - мікрооперації, під час яких виконуються деякі елементарні дії.

Для визначення тимчасових співвідношень між різними етапами операцій використовується поняття машинного такту. Машинній такт визначається як інтервал часу, під час якого виконується одна або декілька одночасних мікрооперацій.

Спрощена структура процесора представлена на рис.3.4. До складу процесора входять наступні блоки.

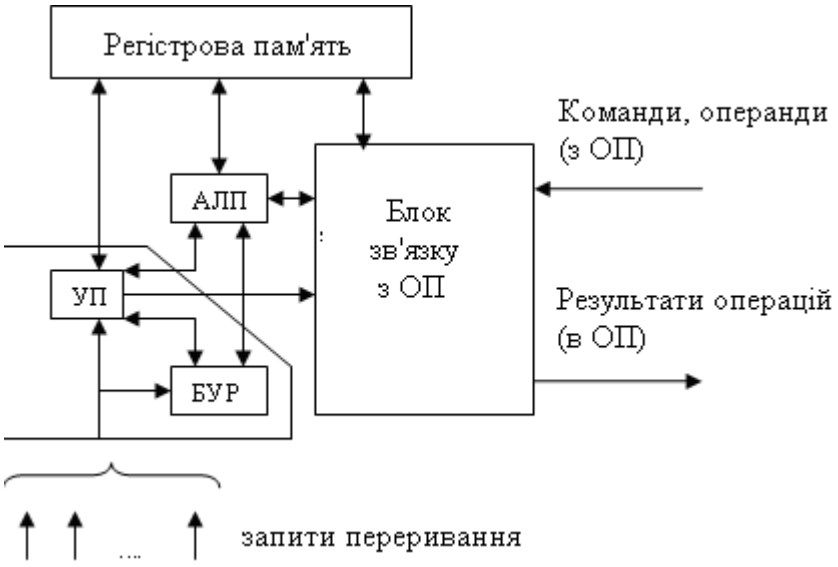


Рис.3.3. Структура процесора.

АЛП - *арифметико-логічний пристрій* процесора виконує логічні і арифметичні операції над даними. В процесорі може бути одне універсальне АЛП або декілька спеціалізованих для окремих видів операцій;

УП – *керуючий пристрій* виробляє послідовність керуючих сигналів, що ініціюють виконання відповідної послідовності мікроопераций, що забезпечують виконання поточної команди. Розрізняють два види пристроїв управління: з «жорсткою» логікою і з логікою, що зберігається в пам'яті.

БУР – *блок керуючих регістрів,* призначений для тимчасового зберігання керуючої інформації. Він містить регістри і лічильники, що беруть участь в управлінні обчислювальним процесом: регістри, що зберігають інформацію про стан процесора, регістр-лічильник адреси команд (лічильник команд), лічильники тактів, регістр запитів переривання і таке інше.

Регістрова пам'ять - блок пам'яті невеликої ємкості, але швидший, ніж ОП. Регістри цього блоку вказують в командах шляхом укороченої регістрової адресації. Вони служать для зберігання операндів команд, у якості акумулятора, базових та індексних регістрів, покажчика стека.

Блок зв'язку (інтерфейс процесора) організовує обмін інформацією процесора з оперативною пам'яттю та захист ділянок ОП від недозволених даній програмі звернень, а також зв'язок процесора з периферійними приладами та зовнішнім устаткуванням.

Блок контролю і діагностики служить для виявлення збоїв і відмов в апаратурі процесора, відновлення роботи програми після збоїв і пошуку місця несправності при відмовах.

В складних КС розрізняють центральний і периферійний процесори. Центральній процесор управляє дією декількох периферійних. Периферійні процесори можуть викликати один одного або ж через послідовність операцій викликати самих себе.

Процесор, реалізований у вигляді одного кристала, називається мікропроцесором.

Всередині ВС можуть виникати події, що вимагають негайних реакцій. Для їх обробки передбачається процес переривання: система перериває обробку поточного завдання і переходить до виконання спеціально призначеної для даної ситуації процедури, після закінчення якої повертається до виконання перерваної програми. Моменти виникнення подій, що вимагають переривання, невідомі. Сигнал, що оповіщає про переривання, називається запитом переривання.

Програмні і апаратні засоби для обслуговування переривань називаються системою переривань. Існують системи з послідовним переглядянням запитів і з обслуговуванням в порядку пріоритетів. Для оцінки системи переривань використовують параметри:

t_p - реакція - час між появою запиту і початком виконання перерваної програми;

$t_з$ - час запам'ятовування стану перерваної програми;

$t_в$ - час відновлення стану перерваної програми;

t_p - час виконання програми, що перервала виконання.

3.2.2. Суперскалярні процесори.

Процесори сімейства Pentium.

Процесор Pentium в порівнянні зі своїми попередниками має цілий ряд покращених характеристик. Головними його особливостями є:

- - двопотокова суперскалярна організація, що допускає паралельне виконання пари простих команд;
- наявність двох незалежних двоканальних множинно-асоціативних кешей для команд і для даних, що забезпечують вибірку даних для двох операцій в кожному такті;
- - динамічне прогнозування переходів;
- - конвеєрна організація пристрою плаваючої крапки з 8 рівнями;

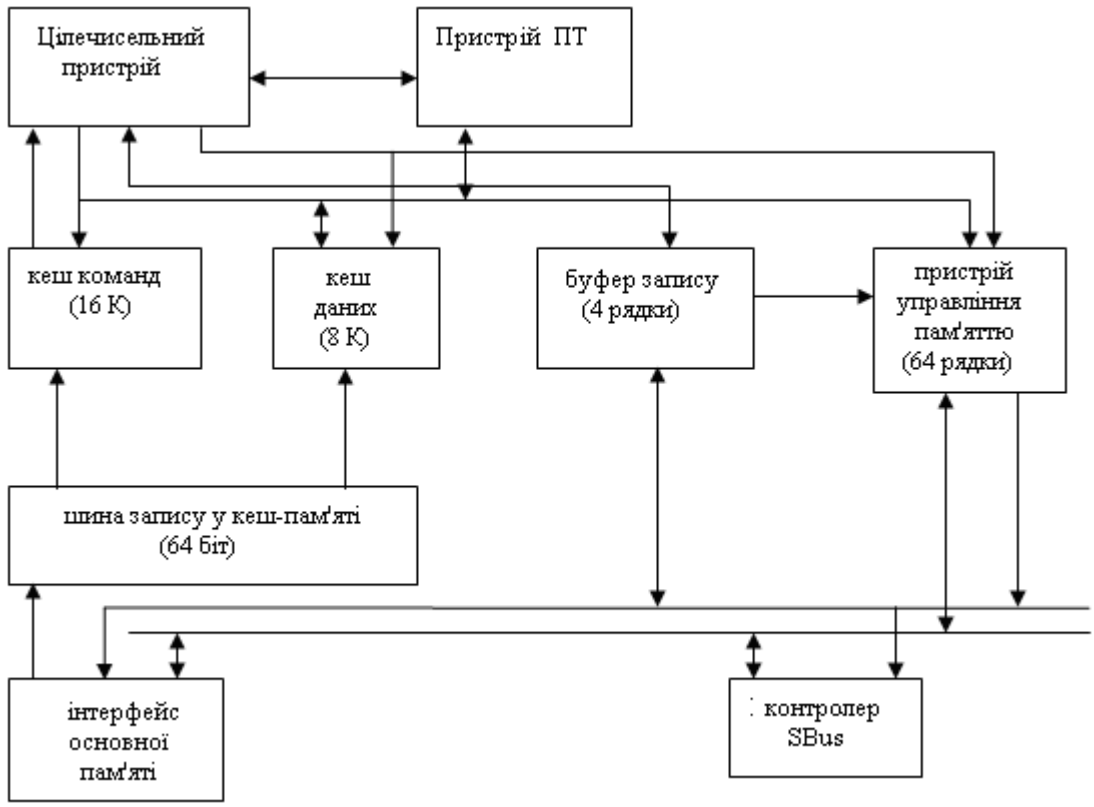


Рис.3.4. Структура процесора Pentium.

- - двійкова сумісність з існуючими процесорами сімейства 80x86.

Блок-схему процесора Pentium представлено на рис.3.4. Мікроархітектура цього процесора базується на ідеї суперскалярної обробки. Основні команди розподіляються по двох незалежних виконавчих пристроях (конвеєрам U і V). Конвеєр U може виконувати будь-які команди сімейства x86, включаючи цілочисельні команди і команди з плаваючою крапкою. Конвеєр V призначений для виконання простих цілочисельних команд і деяких команд з плаваючою крапкою. Команди можуть направлятися в кожен з цих пристроїв одночасно, причому при видачі пристроєм керування в одному такті пари команд складніша команда поступає в конвеєр U, а менш складна - в конвеєр V. Така попарна видача команд можлива, правда, лише для обмеженої підмножини цілочисельних команд. Команди арифметики з плаваючою крапкою не можуть запускатися в парі з цілочисельними командами. Одночасна видача двох команд можлива лише за відсутності залежностей по регістрах. При зупинці команди з будь-якої причини в одному конвеєрі, як правило, зупиняється і другий конвеєр.

Кеш-пам'ять процесора:

- роздільна для команд і даних і має ємкість по 8 Кбайт;
- за один такт з кожної кеш-пам'яті може бути прочитано два слова;

- побудована за принципом двократного розшарування, що забезпечує одночасне зчитування двох слів, що належать одному рядку кеш-пам'яті;
- зберігає відразу три копії тегів;
- для підвищення ефективності кеш-пам'яті в процесорі застосовується 64-бітова зовнішня шина даних.

В процесорі передбачено механізм динамічного прогнозування напряму переходів. З цією метою на кристалі розміщена невелика кеш-память, яка називається буфером цільових адрес переходів і дві незалежні пари буферів попередньої вибірки команд (по два 32-бітові буфери на кожен конвеєр). Буфер цільових адрес переходів зберігає адреси команд, які знаходяться в буферах попередньої вибірки. Робота буферів попередньої вибірки організована таким чином, що в кожен момент часу здійснюється вибірка команд лише в один з буферів відповідної парі. При виявленні в потоці команд операції переходу обчислений адрес переходу порівнюється з адресами, що зберігаються в буфері.

У разі збігу передбачається, що перехід буде виконаний та дозволяється робота іншого буфера попередньої вибірки, який починає видавати команди для виконання у відповідний конвеєр. При розбіжності вважається, що перехід виконуватися не буде і буфер попередньої вибірки не перемикається, продовжуючи звичайний порядок видачі команд. Це дозволяє уникнути простоїв конвеєрів при правильному прогнозі напряму переходу. Остаточне рішення про напрям переходу приймається на підставі аналізу коду умовного переходу. При неправильно зробленому прогнозі вміст конвеєрів анулюється і видача команд починається з необхідної адреси. Неправильний прогноз приводить до призупинення роботи конвеєрів на 3-4 такти.

Подальший розвиток архітектури йде у напрямку на використання технічних рішень, RISC- процесорів:

- виконання команд не за вказаною програмою послідовностю, що усуває в багатьох випадках призупинення конвеєрів череє очікування операндів;
- використання методики перейменування регістрів, що дозволяє збільшувати ефективний розмір регістрового файлу (мала кількість регістрів - одне з найвужчих місць архітектури x86);
- розширення суперскалярних можливостей по відношенню до процесора Pentium, в якому забезпечується одночасна видача лише двох команд з досить жорсткими обмеженнями на їх комбінації.

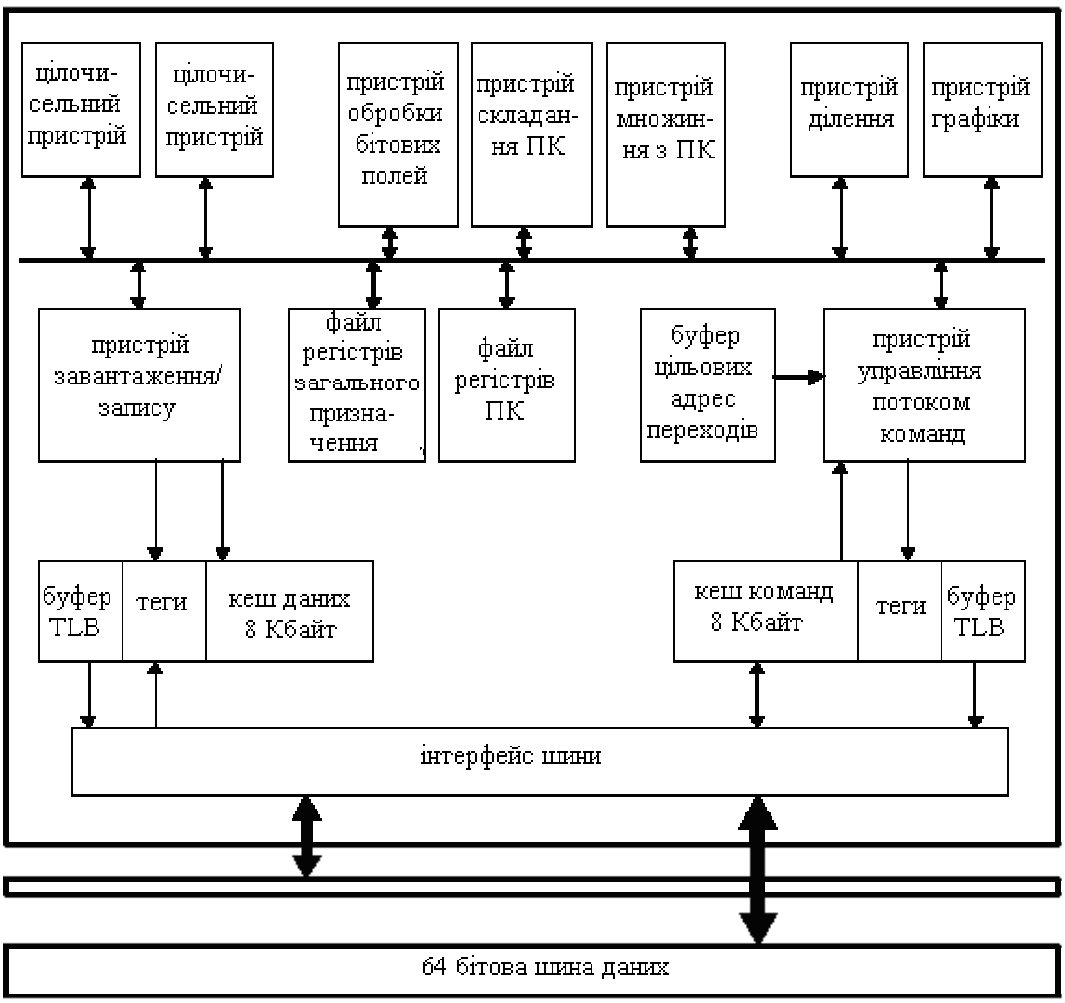


Рис.3.5. Структурна схема процессору Motorola 88110.

Процесор MC88110 компанії Motorola.

Процесор 88110 відноситься до розряду суперскалярних RISC-процесорів та містить 1.3 мільйона логічних вентилів. Основні особливості цього процесора:

- використання принципів суперскалярної обробки;
- два восьмипортових регістрових файла;
- десять незалежних виконавчих пристроїв;
- великі за об'ємом внутрішні кеші;
- широкі магістралі даних;
- можливості завершення команд не в порядку їх отримання для виконання.

Блок-схему процесора представлено на рис. 3.5.

До основних внутрішніх устроїв відносяться:

- шина операндів (в реалізації це шість 80-бітових шин), що сполучає регістрові файли і виконавчі пристрої та є центральною частиною цієї архітектури;

- 10 виконавчих пристроїв, які працюють одночасно і незалежно та два регістрові файли. Файл регістрів загального призначення має 32-бітову організацію. Розширені регістри плаваючої крапки мають 80-бітову організацію. Ці регістрові файли забезпечено шістьма портами читання і двома портами запису кожен;
- зовнішня шина процесора має окремі лінії даних (64 біт) та адреси (32 біт), що дозволяє реалізувати швидкі групові операції перезавантаження внутрішньої кеш-пам'яті. Зовнішня шина має також спеціальні сигнали управління, що забезпечують апаратну підтримку когерентності кеш-пам'яті в мультипроцесорних конфігураціях;
- два двоканальні множинно-асоціативні кеші ємністю по 8 Кбайт (для команд і для даних). Всі операції по перезавантаженню кеш-пам’яті виконуються в режимі групової пересилки даних, при цьому першим пересилається необхідне слово. Когерентність кешів даних забезпечується апаратним протоколом спостереження за шиною з чотирма станами. Для збільшення продуктивності в кеш-пам'яті даних застосовується стратегія затриманого зворотного копіювання.

Суперскалярна архітектура процесора базується на реалізації можливості *завершення команд не в порядку їх отримання для виконання*, що дозволяє істотно збільшити продуктивність, проте призводить до проблем організації точного переривання. Ця проблема вирішується в процесорі 88110 за допомогою так званого буфера історії, який зберігає старі значення регістрів при виконанні та завершенні операцій не в установленому програмою порядку, і дозволяє апаратно відновити необхідний стан в разі переривання.

У процесорі передбачено кілька способів прискорення обробки умовних переходів. Один з них, *прогнозування напрямку переходу*, дозволяє компілятору повідомити процесору переважний напрямок переходу. Для виконуваних переходів використовується буфер цільових адрес переходу ємністю 32 рядка, що дозволяє швидко вибрати дві команди за цільовою адресою переходу. Механізм передбачення напрямку переходів дозволяє одночасно виконувати ці команди й оцінювати умову переходу. Для передбаченого напрямку переходу дозволено *спекулятивне (умовне) виконання команд*. Якщо напрямок переходу передбачено невірно, початковий стан процесора відновлюється за допомогою буфера історії. Виконання програми в цьому випадку буде продовжено з «правильної» команди.

У кожному такті процесор може видавати на виконання дві команди. У більшості випадків видача команд здійснюється у порядку, встановленому програмою. У пристрої завантаження/запису реалізований буфер завантаження FIFO на чотири рядки і три станції резервування операцій запису, що дозволяє мати в кожен момент часу до 4 *відкладених команд*

завантаження і до трьох команд запису. Виконання цих команд всередині пристрою може переупорядковуватися для забезпечення більшої ефективності.

3.2.3 Багатоядерні процесори.

Ідеологія побудови багатоядерних процесорів відповідає багатопроцесорним системам. За доступом до пам’яті багатоядерні процесори поділяють наступним чином.

1. *SMP-системи* (Symmetrical Multi Processor systems). В подібній системі (рис.3.6) всі процесори мають рівноправний доступ до загальної оперативної пам'яті. Створювати подібні системи украй важко: 2-4 процесори - практична межа для SMP-систем. Це пов’язано з дуже обмеженою полосою пропускання на шині.

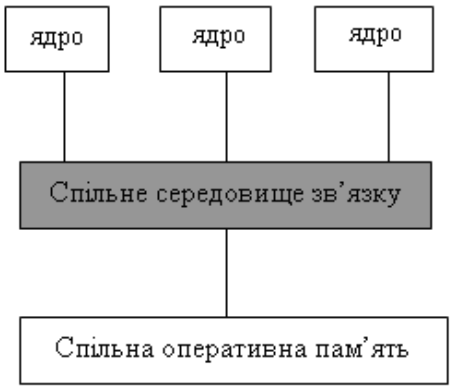


Рис.3.6. Структура багатоядерного процесору, що базується на SMP-архітектурі.

2. NUMA-системи (Non-Uniform Memory Access systems). В даній архітектурі (рис.3.7) доступ до пам'яті є неоднорідним при цьому один її фрагмент є швидшим за інший. В системі при цьому створюються своєрідні «острівці» зі своєю, швидкою «локальною» оперативною пам'яттю, сполучені відносно повільними лініями зв'язку. Писати програми для таких систем складніше, оскільки необхідно враховувати неоднорідності пам'яті

Рис.3.7. Структура багатоядерного процесору,
що базується на NUMA-архітектурі.

3. Останній тип багатопроцесорних систем - кластери. Беремо деяку кількість «майже самостійних» комп'ютерів (вузли кластера або «ноди») і об'єднуємо їх швидкодіючими лініями зв'язку. Загальної пам'яті в цій схемі може і не бути взагалі. Ця система дуже незручна для програмістів та істотно обмежена сфера її вживання. Проте побудова таких систем є дешевою. На сьогодні архітектура SMP переважає в процесорах Intel, NUMA – в процесорах від AMD, IBM та Sun, кластери застосовуються переважно в суперкомп'ютерах.

В архітектурі, якої дотримується компанія *Intel* є один чіпсет, до якого підключається вся оперативна пам'ять та одна процесорна шина, до якої підключені всі процесори. Загальна схемотехніка в цих ядрах не передбачена. Такий централізований підхід відрізняється відносною простотою, а також тим, що в ньому кожен компонент комп'ютера виходить вузькоспеціалізованим та його можна модернізувати незалежно від інших.

Архітектура AMD K8 концептуально інша: в ній немає якогось виділеного центру. Кожен з процесорів архітектури AMD64 є незалежною та «самодостатньою» одиницею: в нього інтегровано практично всю функціональність північного моста. Однак AMD свою архітектуру називає SUMA: і не SMP і не NUMA. Відмінність від NUMA досягається наступним шляхом. В кожен процесор інтегрується контролер «локальної» оперативної пам'яті. Звернення до пам'яті «чужих» процесорів відбуваються по шині HyperTransport, причому робиться ця «переадресація» запитів абсолютно прозоро для власне

обчислювального ядра процесора - її здійснює вбудований в Northbridge комутатор (CrossBar). Цей же самий CrossBar забезпечує «автоматичну» маршрутизацію повідомлень від периферійних пристроїв і інших процесорів, включаючи обслуговування «чужих» запитів до оперативної пам'яті HyperTransport. Модель пам'яті виходить неоднорідною (NUMA), але відмінності в швидкості «своїх» і «чужих» ділянок оперативної пам'яті виходять відносно невеликими.

Загальними проблемами при побудові багатоядерних процесорів за будь якою з перерахованих схем є наступні.

1. *Латентність (час відгуку) оперативної пам'яті* в обох схемах залишається великою, особливо для NUMA. Але для зменшення цього ефекту використовують кеш-пам'ять великих розмірів та механізми апаратного і програмного передчасного вибирання з пам'яті.

2. Проблема масштабованості - коли використання декількох процесорів не наводить до очікуваного приросту продуктивності. За законом Амдаля легко отримати наступні практичні цифри: якщо одиночний процесор 20% свого часу простоював, чекаючи даних з оперативної пам'яті, то процесор з двома ядрами простоюватиме 33% часу, а з чотирма - 50%.

3. Проблема загальної шини: необхідно забезпечити досить швидку передачу отриманих з пам'яті даних до процесора. Чим більше процесорів мі поміщаємо на системну шину, тим складніше забезпечити безпомилкову передачу по ній даних (зростає електричне навантаження, ускладнюється розводка). Але в неоднорідній AMP-шній SUMA-архітектурі цієї проблеми взагалі немає. Ядра один одному практично не заважають - їх обмежує лише пропускна спроможність оперативної пам'яті і ведучих в «зовнішній світ» лінків HyperTransport.

3.2.3.1 Основні технології підвищення продуктивності в багатоядерних процесорах.

1. Технологія HyperThreading.

Hyper Threading є логічним продовженням NetBurst. У середньому при виконанні коду, типового для набору команд IA-32, реально використовується лише 35% виконавчих ресурсів процесора, а 65% виконавчих ресурсів процесора простоюють. Саме цю ідею і реалізує технологія Hyper-Threading, підключаючи незадіяні ресурси процесора до виконання паралельного завдання. Дана технологія є чимось середнім між багатопотоковою обробкою, реалізованою в мультипроцесорних системах, і паралелізмом на рівні інструкцій,

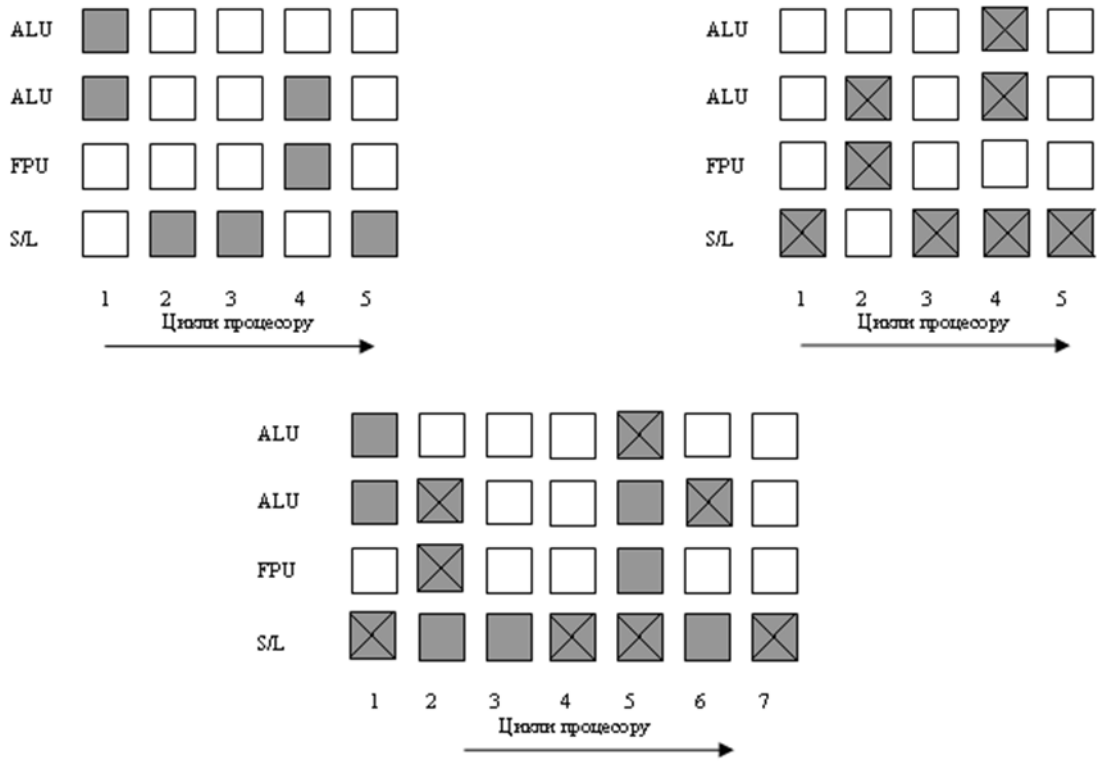


Рис.3.8. Приклад роботи технології HyperThreading.

реалізованому в однопроцесорних системах. Hyper-Threading дозволяє організувати два логічні процесори в одному фізичному. (рис.3.8)

Сірими квадратами замальовані команди одного конвеєру процесора, що працюють в даний такт часу. Виконання двох потоків з використанням HyperThreading зайняло не 5, а 7 тактів. З переходом на C2D, Intel відмовляється від Hyper-Threading, що знов з'являється в Core i7.

Intel також запропонувала технологію, що є зворотною до Hyper-Threadingr. Ця технологія об'єднує обчислювальні ресурси двох ядер процесора для обробки одного потоку, що повинно прискорювати роботу програм, які не відчувають вигоди від переходу на багатоядерну платформу. Ця технологія Intel має назву EDAT.

2. Технологія HyperTransport.

Шина HyperTransport (HT) - це двонаправлена послідовно/паралельна комп'ютерна шина, з високою пропускною спроможністю і малими затримками; це шина нового покоління спеціально для чипів з інтегрованим контролером пам'яті. В багатопроцесорних системах на основі AMD Opteron у кожного ядра є своя локальна пам'ять. Локальна пам'ять "швидка", а

пам'ять сусіда - "повільна". Нова шина покликана забезпечити пропускну спроможність не меншу, ніж в оперативної пам'яті і мінімальні затримки на передачу даних і повідомлень. SUMA - Slightly Uniform Memory Architecture, тобто "майже однорідна" архітектура пам'яті.

Будь-які дані, що передаються по шині, упаковуються в пакети стандартного вигляду. HyperTransport підтримує автоматичне визначення ширини шини, від 2-х бітових ліній до 32-х бітових ліній. Базова тактова частота шини HT - 200 Мгц (тобто частота передачі даних - 400 Мгц).

Відмінність HT, від інших послідовних пакетних протоколів в тому, що передачу одного пакету, можливо перервати і передати інший пакет – який є більш терміновим. При досить широкій шині типова затримка передачі повідомлення в HT складає 1-2 такти.

Ця шина використовується окрім AMD Athlon 64/Opteron ще і у новітніх процесорах Apple G5 (IBM) і супутніх чіпсетах, в процесорах Transmeta, у мережевому обладнанні (Cisco, Broadcom), в чіпсетах Nvidia і ALi/ULi.

Наступні технології використовуються в Intel Core Microarchitecture.

Продуктивність визначається як множення тактової частоти процесора на величину, що визначає кількість інструкції, виконуваних CPU за один такт. Таким чином є дві основні дороги для збільшення швидкодії: нарощування тактової частоти та збільшення числа інструкцій, що виконуються за один такт.

Крім того є і ще один метод для підвищення швидкості процесорів — зменшення числа операцій, необхідних для обробки одних і тих же об'ємів даних. Гарною ілюстрацією прогресу в цьому напрямі можна вважавати впровадження наборів SIMD інструкцій SSE, SSE2 і SSE3, що дозволяють виконувати векторні операції.

Що ж до енергоспоживання, то воно представляється як множення тактової частоти процесора, квадрата напруги, при якій функціонує процесорне ядро і деякої константи "динамічна ємкість", що визначається мікроархітектурою CPU і залежить від числа транзисторів та їх активності під час роботи процесора.

Для оптимізації мікроархітектури з точки зору співвідношення продуктивності і енергоспоживання розробники повинні фокусуватися на встановленні балансу між кількістю інструкцій, що виконуються. процесором за такт і динамічною ємкістю.

Merom, Conroe and Woodcrest Block Diagram

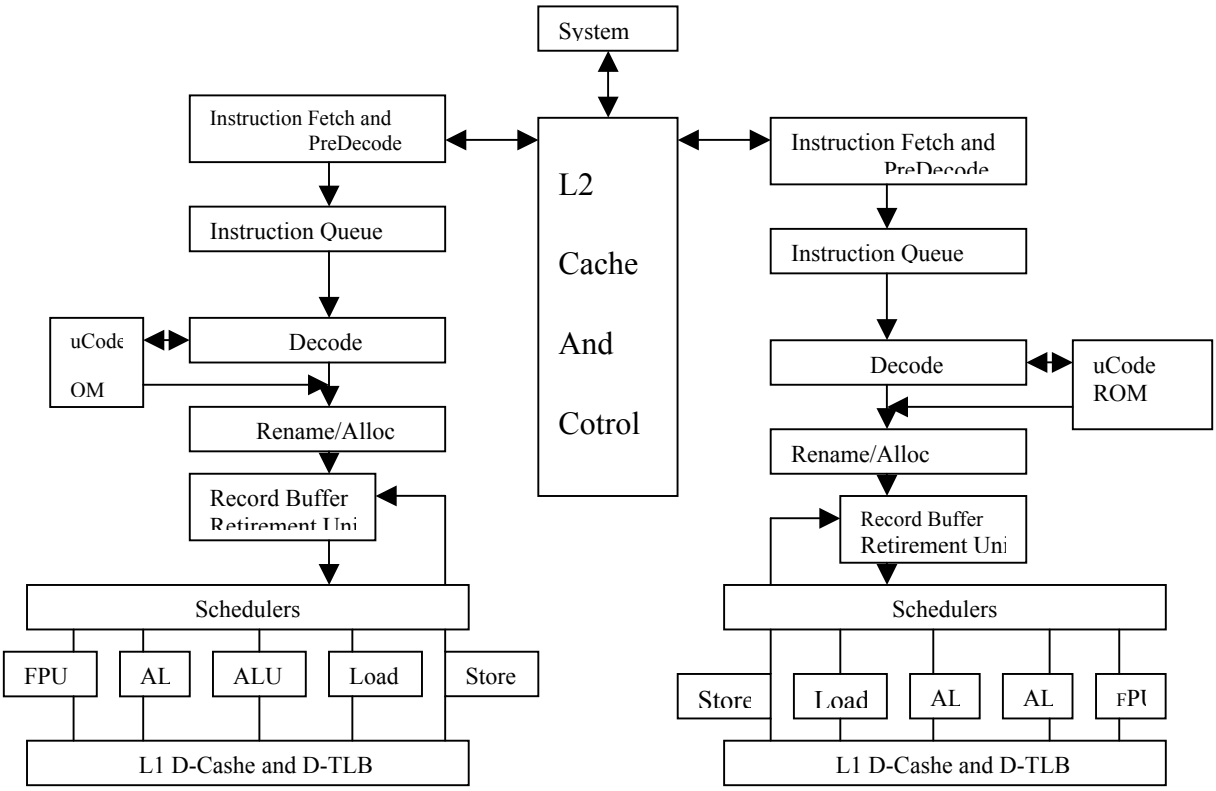


Рис. 3.9 Структура двоядерного процесору мікроархітектури Core.

Еволюційній ряд Core просувався в бік покращення всіх цих характеристик. Еволюційній ряд Core складають: Pentium Pro, Pentium II, Pentium III, Pentium M, Core, Core 2.

Остання верся Intel Core 2 – ядро двоядерних процесорів. В Core 2 Duo, як і у першому Core Duo, використовується загальний кеш другого рівня, до якого обидва ядра мають рівноправний доступ.

Структура двоядерних процесорів мікроархітектури Core наведено на рис.3.9.

Процесори, що базуються на Intel Core Microarchitecture, зможуть обробляти до чотирьох інструкцій за такт. Довжина виконавчого конвеєра процесорів з мікроархітектурою Core складає 14 стадій. З точки зору продуктивності короткий конвеєр є швидше плюсом, чим мінусом.

3. Intel Wide Dynamic Execution, що є основою мікроархітектури Core. Основною її властивістю є поглиблена суперскалярність, а саме:

–спекулятивне та позачергове виконання команд;

–поглиблений аналіз коду з покращенням алгоритмів передбачення переходів;

–додавання декодерів команд та виконавчих пристроїв, що дозволяє виконувати до 4 команд за такт (процесори AMD та попередні Intel можуть обробляти не більше трьох інструкцій за такт).

4. Технологія macrofusion.

Попередники Core, процесори Pentium M, володіли надзвичайно цікавою технологією micro-ops fusion, направленою на зниження "Накладних витрат" при виконанні деяких x86 команд. Суть технології у наступному. У випадку якщо x86 команда розпадається на залежні один від одного мікроінструкції, декодер здійснює їх прив'язку один до одного. Такі послідовності мікроінструкцій, "склеєні" технологією micro-ops fusion для виконання процесором в певному порядку, представляються процесором на всіх етапах, окрім власне виконань, однією командою. Це дозволяє уникнути непотрібних простоїв процесора в разі, якщо зв'язані мікроінструкції виявляються відірваними один від одного в результаті роботи алгоритмів позачергового виконання.

На додаток до вельми вдалої технології micro-ops fusion, мікроархітектура Core отримала технологію macrofusion. Дана технологія направлена на збільшення числа виконуваних за такт команд і полягає в тому, що ряд пар зв'язаних між собою послідовних x86 інструкцій, таких як, наприклад, порівняння з наступним за ним умовним переходом, представляються усередині процесора однією мікроінструкцією. Така мікроінструкція розглядається та виконується на виконавчих прибудовах як одна команда, що дозволяє досягти як збільшення темпу виконання коду, так і деякої економії енергії.

5. *Технологія Intel Smart Memory Access* - підвищує продуктивність системи шляхом зниження затримок при доступі до пам'яті і таким чином оптимізує використання доступної пропускної спроможності, завдяки чому процесор отримує дані тоді, коли вони потрібні.

4. *Технологія Intel Advanced Smart Cache.*

Загальна кеш-пам'ять 2-го рівня скорочує енергоспоживання, зводячи до мінімуму об'єм трафіку в підсистемі пам'яті і підвищує продуктивність системи, забезпечуючи одному з ядер доступ до всієї кеш-пам'яті при простої іншого ядра. Доступ до всього об'єму L2 кеша може отримати будь-яке з ядер процесора Core. Це, зокрема, означає і те, що коли одне з ядер не діє, друге отримує в своє повне розпорядження весь об'єм кеш-пам'яті. Якщо ж одночасно працюють два процесорні ядра, то кеш ділиться між ними пропорційно в залежності від частоти звернень кожного ядра до оперативної пам'яті. Більш того, якщо два ядра працюють синхронно з одними і тими ж даними, то вони зберігаються в L2 кеші лише одноразово.

5. *Технологія Intel Advanced Digital Media Boost* - подвоює швидкість виконання команд, що часто використовуються в мультимедійних і графічних застосуваннях, що дозволило прискорити роботу з SSE-інструкціями у два рази.

6. *Технологія Intel Intelligent Power Capability* – це набір технологій, направлених на зниження енергоспоживання і тепловиділення з метою оптимізації параметра "продуктивність на ват". Процесори мають можливість інтерактивного відключення власних підсистем (не ядер в цілому), які не використовуються в даний момент. Кожне з процесорних ядер поділене на велику кількість блоків і внутрішніх шин, живлення якими управляється окремо за допомогою спеціалізованих додаткових логічних схем. Головною особливістю є те, що така робота не спричиняє за собою збільшення часу відгуку процесора на зовнішні дії, викликане необхідністю переводити відключені блоки у функціональний стан.

7. *Intel 64 Technology* — забезпечує підтримку 64-розрядних обчислень, надаючи процесору доступ до більшого об'єму пам'яті.

В табл.3.3 наводиться коротке порівняння характеристик процесорів Intel та AMD останнього покоління.

Табл.3.3.Порівняння характеристик процесорів Intel Core та AMD K8.

	INTEL Core	AMD K8
L1 кеш даних	32 Кбайт	64 Кбайт
L1 кеш інструкцій	32 Кбайт	64 Кбайт
Латентність кеша L1	3 циклу	3 циклу
Асоціативність L1 кеша	8-way	2-way
Розмір L1 TLB	Інструкції-128 входжень	Інструкції-32 входжень
Максимальній розмір L2 кеша	Дані-256 входжень	Дані-32 входжень
Латентність L2 кеша	4 Мбайта на два ядра	1 Мбайт на кожне ядро
Асоціативність L2	14 циклів	12 циклів
Ширіна шини L2 кеша	16- way	16- way
Розмір L TLB	256 біт	128 біт
Довжіна конвеєра	-	-
Число x86 декодерів	14 стадій	12 стадій
Цілочисельні виконавчі пристрої	3ALU+2AGU	3ALU+3AGU
Load/Store прибудую	2(1 Load+1 Store)	2
FP виконавчі пристрої	FADD+FMUL+FLOAD+FSTORE	FADD+FMUL+FSTORE
SSE виконавчі пристрої	3(128-бітові)	3(64-бітові)

3. Організація підсистем введення-виведення

3.3.1 Системні і локальні шини

У обчислювальній системі, що складається з цілої множини компонент необхідний механізм їх взаємодії. Одним з простих механізмів, що дозволяють здійснити таку взаємодію, є єдина центральна шина, до якої приєднані всі підсистеми.

Переваги такої організації:

- низька вартість;
- універсальність;
- простота підключення нових пристроїв.

Недоліки:

- створює «вузьке місце» в системі, якщо весь потік введення/виведення повинен проходити через центральну шину.

Таким чином, для систем з великим трафіком введення/виведення і високопродуктивних систем актуальним є створення системи, що складається з декількох шин, яка обслуговує всі запити введення/виведення.

Основними параметрами при створенні шин є: швидкодія, кількість пристроїв, що можна приєднати до шини, фізична довжина та пропускна здатність. Для підвищення швидкодії у високопродуктивних системах використовують декілька взаємозв'язаних шин, кожна з яких спрощує взаємодію однієї або декількох підсистем, а також забезпечує високу пропускну спроможність і фізичну надмірність (для підвищення відмовостійкості).

Традиційно шини поділяють на шини, що забезпечують організацію зв'язку процесора з пам'яттю, і шини введення/виведення. Шини введення/виведення можуть мати велику довжину, підтримувати приєднання багатьох типів пристроїв, і зазвичай слідують одному з шинних стандартів. Шини процесор-пам'ять, з іншого боку, порівняно короткі, зазвичай високошвидкісні і відповідають організації системи пам'яті для забезпечення максимальної пропускної спроможності каналу пам'ять-процесор. На етапі розробки системи, для шини процесор-пам'ять заздалегідь відомі всі типи і параметри пристроїв, які повинні з'єднуватися між собою, тоді як розробник шини введення/виведення повинен мати справу з пристроями, що розрізняються по затримці і пропускній спроможності.

Якщо комп'ютер має загальну шину процесор-пам'ять і введення/виведення, то вона називається системною. Такі шини застосовуються в персональних комп'ютерах. Подальшим

їх розвитком є локальні шини, що дозволяють зберігати баланс вартості і продуктивності. Локальна шина фізично виходить своїми контактами безпосередньо на шини процесора і об'єднує: процесор, пам'ять, буфери системної шини і її контролер.

Обмін порцією інформації по шині називається транзакцією. Вони поділяються на транзакції запису і читання. Розглянемо типовий приклад транзакції.

Шинна транзакція включає дві частини: посилку адреси і прийом (або посилку) даних. У транзакції типу "читання" по шині спочатку посилається в пам'ять адреса разом з відповідними сигналами управління, що відображають читання. Пам'ять відповідає, повертаючи на шину дані з відповідними сигналами управління. Транзакція типу "запис" вимагає, аби ЦП або пристрій введення/виведення послав в пам'ять адресу та дані і не чекає повернення даних. Зазвичай ЦП вимушений простоювати під час інтервалу між посилкою адреси і отриманням даних при виконанні читання, але часто він не чекає завершення операції при записі даних в пам'ять.

Розробка шини пов'язана з реалізацією ряду додаткових можливостей (таблиця. 3.4).

Вирішення про вибір тієї або іншої можливості залежить від цільових параметрів вартості і продуктивності. Перші три можливості є очевидними: роздільні лінії адреси і даних, ширші (що мають велику розрядність) шини даних і режим групових пересилок (пересилки декількох слів) дають збільшення продуктивності за рахунок збільшення вартості.

Табл.3.4.Основні можливості при проектуванні шин.

Можливість	Висока продуктивність	Низька вартість
Загальна розрядність шини	Окремі лінії адреси і даних	Мультиплексування ліній адреси і даних
Ширина (розрядність) даних	Чим ширше, тим швидше (наприклад, 32 біт)	Чим вужче, тим дешевше (наприклад, 8 біт)
Розмір пересилки	Пересилка декількох слів має менші накладні витрати	Пересилка одного слова дешевша
Головні пристрої шини	Декілька (потрібний арбітраж)	Одне (арбітраж не потрібний)
Розщеплені транзакції	Так - окремі пакети запиту і відповіді дають більшу смугу пропускання (потрібно декілька головних пристроїв)	Немає з'єднання, що продовжується, дешевше і має меншу затримку
Тип синхронізації	Синхронні	Асинхронні

Головний пристрій шини - це пристрій, який може ініціювати транзакції читання або запису. ЦП, наприклад, завжди є головним пристроєм шини. Шина має декілька головних пристроїв, якщо є декілька ЦП або коли пристрої введення/виведення можуть ініціювати транзакції на шині. Якщо є декілька таких пристроїв, то потрібна схема арбітражу, аби вирішити, хто наступний захопить шину. Арбітраж часто заснований або на схемі з фіксованим пріоритетом, або на більш "справедливій" схемі, яка випадковим чином вибирає, який головний пристрій захопить шину.

В даний час використовуються два типи шин, що відрізняються способом комутації: шини з *комутацією ланцюгів* і шини з *комутацією пакетів*. Шина з комутацією пакетів за наявності декількох головних пристроїв шини забезпечує значно більшу пропускну спроможність в порівнянні з шиною з комутацією ланцюгів за рахунок розділення транзакції на дві логічні частини: запиту шини і відповіді. Така методика отримала назву "розщеплювання" транзакцій. У деяких системах така можливість називається конвеєрною шиною. Транзакція читання розбивається на транзакцію запиту читання, яка містить адресу, і транзакцію відповіді пам'яті, яка містить дані. Кожна транзакція тепер має бути помічена (тегирована) відповідним чином, аби ЦП і пам'ять могли розрізнити етапи транзакцій.

Шина з комутацією ланцюгів не робить розщеплювання транзакцій: будь-яка транзакція на ній є неділимою операцією. Головний пристрій запитує шину, після арбітражу поміщає на неї адресу і блокує шину до закінчення обслуговування запиту. Велика частина часу обслуговування при цьому витрачається не на виконання операцій на шині, а, наприклад, на затримку вибірки з пам'яті. Таким чином, в шинах з комутацією ланцюгів цей час просто втрачається. Розщеплені транзакції роблять шину доступною для інших головних пристроїв, поки пам'ять читає слово за запитаною адресою. Це також означає, що ЦП повинен боротися за шину для посилки даних, а пам'ять повинна боротися за шину, аби повернути дані. Таким чином, шина з розщеплюванням транзакцій має вищу пропускну спроможність, але зазвичай вона має і більшу затримку, ніж шина, яка захоплюється на весь час виконання транзакції. Транзакція називається розщепленою, оскільки довільна кількість інших пакетів або транзакцій може використовувати шину між запитом і відповіддю.

Тип синхронізації визначає: чи є шина *синхронною* або *асинхронною*. Якщо шина синхронна, то вона включає сигнали синхронізації, які передаються по лініях управління шини, і фіксований протокол, що визначає розташування сигналів адреси і даних відносно сигналів синхронізації. Оскільки практично жодної додаткової логіки не потрібно для того, щоб вирішити, що робити в наступний момент часу, ці шини можуть бути і швидкими, і дешевими. Проте така шина має суттєвий недолік: всі дії на шині повинні відбуватися з

однією і тією ж частотою синхронізації, тому через проблеми перекосу синхросигналів, синхронні шини не можуть бути довгими. Зазвичай шини процесор-пам'ять синхронні.

Асинхронна шина, з іншого боку, не тактується. Замість цього використовується старт-стопний режим передачі і протокол "рукостискання" (handshaking) між джерелом і приймачем даних на шині. Ця схема дозволяє набагато простіше пристосувати широку різноманітність пристроїв і подовжити шину без занепокоєння про перекіс сигналів синхронізації і про систему синхронізації. Якщо може використовуватися синхронна шина, то вона зазвичай швидше, ніж асинхронна, через відсутність накладних витрат на синхронізацію шини для кожної транзакції. Вибір типа шини (синхронна або асинхронна) визначає не лише пропускну спроможність, але також безпосередньо впливає на ємність системи введення/виведення в термінах фізичної відстані і кількості пристроїв, які можуть бути приєднані до шини. Асинхронні шини у міру зміни технології краще масштабуються. Шини введення/виведення зазвичай асинхронні.

3.3.4 Конвеєрний режим шини

Широке використання конвеєризації в мікросхемах пам'яті, про яку йшла мова раніше, привело до істотного збільшення їх ефективності. Цей же підхід дозволяє підвищити продуктивність і шин комп'ютера. На рис.3.14 зображено спрощену схему послідовних тактів роботи шини, що використовує конвеєризацію.

Щоб організувати конвеєр, необхідно виділити в роботі пристрою декілька самостійних етапів і доручити виконання кожного з них окремому вузлу, поєднавши їх роботу в часі. У даній спрощеній ситуації цикл шини розбитий на чотири етапи: фаза арбітражу (А), фаза запиту (З), фаза перевірки і виявлення помилки в запиті (П), фаза передачі запитаних даних (Д). Під час фази арбітражу арбітр за встановленим алгоритмом обирає пристрій, що отримує доступ до шини. Фаза запиту включає формування адреси і необхідних керуючих сигналів. На наступній фазі перевіряється наявність помилок в запиті, і при їх відсутності цикл шини містить останню фазу, пов'язану з передачею запитаних даних. У змальованій на рис.3.14 ситуації передбачається, що запити на виконання операції читання/запису не містять помилок.

команди	такт 1	такт 2	такт 3	такт 4	такт 5	такт 6
1	А	З	П	Д		
2		А	З	П	Д	
3			А	З	П	Д
4				А	З	П
5					А	З
6						А
7						

Рис. 3.10 Спрощена схема конвеєрного режиму.
А-арбітраж, З-запит, П- перевірка помилки, Д-дані.

Розглянемо проходження по шині декількох послідовних операцій читання/запису, кожній з яких відповідає окремий цикл дій на конвеєрі. Для зручності викладу такий цикл називатимемо транзакцією.

1) Перший такт. Перша транзакція проходить фазу арбітражу.

2) Другий такт. Перша транзакція проходить фазу запиту, а друга транзакція - фазу арбітражу.

3) Третій такт. Перша транзакція вийшла на фазу перевірки помилки, друга - на фазу запиту, в цей же час третя транзакція проходить фазу арбітражу.

4) Четвертий такт. Перша транзакція завершується фазою передачі даних, друга транзакція проходить етап перевірки помилок, третя проходить фазу запиту, а арбітраж проходить вже четверта транзакція.

5) П'ятий такт. Друга транзакція вийшла на завершальну фазу передачі даних. Третя проходить перевірку помилок, четверта вступила у фазу запиту, а п'ята проходить арбітраж.

6) Шостий такт. Третя транзакція завершується передачею даних. Четверта проходить перевірку помилок, п'ята знаходиться на фазі запиту, а шоста — на фазі арбітражу.

7) Сьомий такт: і так далі

Як можна бачити з протоколу проходження конвеєра, кожна транзакція окремо триває стільки ж часу, скільки займає неконвеєрний цикл шини. Зате загальна пропускна здатність шини, підрахована за той проміжок часу, що включає декілька транзакцій, виявляється істотно вищою. Збільшення продуктивності може досягти чотирикратного рівня за рахунок того, що в сталому режимі на конвеєрі шини одночасно виконуються чотири різні фази чотирьох різних транзакцій.

3.3.5. Багатошинна архітектура.

Архітектура комп'ютерів, у складі яких є більш ніж одна шина, називається багатошинною.

Коли до складу комп'ютерів стали включати зовнішній кеш, він приєднувався до системної шини, що зв'язує процесор і оперативну пам'ять. Ця архітектура виявилася неефективною, тому що кеш простоював через низьку швидкость шини. У 1997 р. була запропонована архітектура подвійної незалежної шини, або архітектура DIB (від Dual Independent Bus), що мала на увазі наявність двох окремих шин. Одна з них - шина кеша, або шина BSB (від Back Side Bus), - пов'язує процесор із зовнішнім кешем, а друга - системна шина, або шина FSB (від Front Side Bus), - пов'язує процесор з оперативною пам'яттю. Наявність двох незалежних шин, по яких процесор може дістати доступ до даних, що передаються по будь-якій з шин одночасно і паралельно, більш ніж в три рази прискорює роботу зовнішнього кэша.

Отже, сучасні персональні комп'ютери зазвичай містять :

- шину низькошвидкісних зовнішніх пристроїв (клавіатура, миша, принтер);
- шину високошвидкісних зовнішніх пристроїв (магнітні, оптичні диски);
- шину графічного адаптера, для передачі високоякісних кольорових зображень, що передаються на екран дисплея;
- системну шину (шину пам'яті), що зв'язує процесор і оперативну пам'ять;
- шину кешу, що зв'язує процесор і зовнішній кеш.

3.3.3 Дискові масиви та рівні RAID

3.3.2 Основні типи пристроїв введення/виведення

Як правило, периферійні пристрої комп'ютерів поділяються на пристрої введення, пристрої виведення та зовнішні запам'ятовуючі пристрої (які здійснюють як введення даних в машину, так і виведення даних з комп'ютера).

Основною узагальнюючою характеристикою пристроїв введення/виведення може служити швидкість передачі даних (максимальна швидкість, з якою дані можуть передаватися між пристроєм введення/виведення та основною пам'яттю або процесором). У табл. 3.5

представлені основні пристрої введення/виведення, застосовувані в сучасних комп'ютерах, а також вказані орієнтовні швидкості обміну даними, що забезпечуються цими пристроями.

Одним із шляхів підвищення продуктивності введення/виведення є використання паралелізму шляхом об'єднання кількох фізичних дисків в матрицю (групу) з організацією їх роботи аналогічно одному логічному диску. Нажаль, надійність матриці будь-яких пристроїв падає при збільшенні числа пристроїв. Вважаючи інтенсивність відмов постійною, тобто при експоненційному законі розподілу напрацювання на відмову, а також за умови, що відмови незалежні, отримаємо, що середній час безвідмовної роботи (mean time to failure - MTTF) матриці дисків буде дорівнюівати:

MTTF матриці = MTTF одного диска / Кількість дисків в матриці.

Для досягнення підвищеного рівня відмовостійкості доводиться жертвувати пропускною здатністю введення/виведення або ємністю пам'яті. Необхідно використовувати додаткові диски, що містять надлишкову інформацію, що дозволяє відновити вихідні дані при відмові диска. Звідси отримують акронім для надлишкових матриць недорогих дисків RAID (redundant array of inexpensive disks). Існує кілька способів об'єднання дисків RAID. Кожен рівень представляє свій компроміс між пропускною здатністю введення/виведення і ємністю диска, призначеної для зберігання надлишкової інформації.

Коли який-небудь диск відмовляє, передбачається, що протягом короткого інтервалу часу він буде замінений і інформація буде відновлена на новому диску з використанням надлишкової інформації. Цей час називається середнім часом відновлення (mean time to repair - MTTR). Цей показник можна зменшити, якщо в систему входять додаткові диски в якості "гарячого резерву": при відмові диска резервний диск підключається апаратно-програмними засобами. Періодично оператор вручну замінює всі диски, що відмовили. Чотири основні етапи цього процесу полягають у наступному:

- визначення диска, що відмовив;
- усунення відмови без зупину обробки;
- відновлення втрачених даних на резервному диску;
- періодична заміна дисків, що відмовили на нові.

Слід зазначити, що вживання RAID масивів захищає від втрат даних лише при фізичних відмовах жорстких дисків. Використання RAID масивів не може убезпечити Ваші дані у випадку:

- відмови RAID контролера;

- збоїв устаткування;
- збоїв програмних засобів;
- помилкових дій обслуговуючого персоналу;
- помилках або зловмисних дій користувачів;
- вірусних атаках; а також в при виникненні будь-яких інших проблем, що не пов'язані з технічною несправністю накопичувачів.

В RAID-масивах всіх рівнів є загальна характеристика: операційна система сервера, до якого вони підключені, працює з ними як з єдиним логічним диском. Це означає, що можна об'єднати різні RAID-рівні для створення масиву масивів, де фізичні диски замінено RAID-масивами другого рівня, які не обов'язково повинні мати ту ж схему зберігання даних, що і масив першого рівня. Об'єднання масивів дозволяє створити системи зберігання величезної ємкості.

RAID 0. Дисковий масив без відмовостійкої (Striped Disk Array without Fault Tolerance).

Масив дисків без надлишкового зберігання даних. Інформація розбивається на блоки, які записуються на окремі диски, що забезпечує збільшення продуктивності. Даний спосіб зберігання інформації ненадійний (поломка одного диска приводить до втрати всієї інформації), тому рівнем RAID як таким не є. Реалізація цього рівня дуже проста. В основному RAID 0 застосовується в тих сферах, де потрібна швидка передача великого об'єму даних.

Переваги:

- найвища продуктивність в застосуваннях, що вимагають інтенсивної обробки запитів введення/виведення і даних великого об'єму;
- простота реалізації;
- низька вартість.

Недоліки:

- не відмовостійке рішення;
- відмова одного диска спричиняє за собою втрату всіх даних масиву.

RAID 1. Дзеркальні диски (mirroring).

Дисковий масив з дублюванням інформації. У цьому випадку два накопичувачі містять однакову інформацію і є одним логічним диском. Тим самим забезпечується високий рівень збереження даних: при виході з одного диска його функції виконує інший (що абсолютно прозоро для користувача). Крім того, цей рівень подвоює швидкість зчитування інформації, оскільки ця операція може виконуватися одночасно з двох дисків.

Така схема зберігання інформації використовується в основному в тих випадках, де "ціна" безпеки даних набагато вища за вартість реалізації системи зберігання. Але оскільки

ціни на диски весь час знижуються, RAID 1 стає все популярнішим. У серверах середнього рівня, де об'єм інформації, що зберігається, не так великий, його вживання може бути виправдане. RAID 1 простий в реалізації, дозволяє створити відмовостійку систему всього з двох дисків, найбільший його мінус - висока вартість.

Переваги:

- простота реалізації;
- простота відновлення масиву в разі відмови (копіювання).

Недоліки:

- висока вартість - 100% надмірність;
- невисока швидкість передачі даних при записуванні.

RAID 2. Відмовостійкий дисковий масив з використанням коду Хеммінга (Hamming Code ECC).

Один із шляхів досягнення надійності при зниженні втрат ємності пам'яті може бути підказаний організацією основної пам'яті, в якій для виправлення одиночних і виявлення подвійних помилок використовуються надлишкові контрольні розряди.

Схема резервування даних з використанням коду Хеммінга (Hamming code) для корекції помилок - запатентований компанією Thinking Machines. Потік даних розбивається на слова таким чином, що кількість біт в шарі дорівнює кількості дисків і при записі слова кожен окремий біт записується на свій диск. Для кожного слова обчислюється код корекції помилок, який записується на виділені диски для зберігання контрольної інформації. Їх число дорівнює кількості біт в слові контрольної суми.

Наприклад, якщо слово складається з чотирьох біт, то під контрольну інформацію відводиться три диски. RAID 2 - один з небагатьох рівнів, що дозволяє виявляти подвійні помилки і виправляти "на льоту" одиночні. При цьому він є найбільш надлишковим серед всіх рівнів з контролем парності. Така схема зберігання придатна для застосувань, де потрібна передача великого об’єму даних (за рахунок паралельного звернення до дисків), але непридатна для завдань з великою кількістю запитів малого об’єму (за рахунок порівняно великого об’єму операцій, який потрібний для перерозподілу даних). Таким чином, RAID рівня 2 підходить для суперкомп'ютерів, але не підходить для обробки транзакцій. RAID 2 відносно дорогий, але при збільшенні кількості дисків вартість реалізації знижується. Ця схема зберігання даних мало застосовується, оскільки погано справляється з великою кількістю запитів, складна в організації і має незначні переваги перед рівнем RAID 3.

Переваги:

- досить проста реалізація;

- швидка корекція помилок;
- дуже висока швидкість передачі даних;
- при збільшенні кількості дисків накладні витрати зменшуються.

Недоліки:

- низька швидкість обробки запитів;
- велика вартість при малій кількості дисків.

RAID 3. Відмовостійкий масив з паралельною передачею даних і парністю (Parallel Transfer Disks with Parity).

Більшість контрольних дисків, використовуваних в RAID рівня 2, потрібні для визначення положення несправного розряду. Ці диски стають повністю надлишковими, так як більшість контролерів взмозі визначити, коли диск відмовив за допомогою спеціальних сигналів, підтримуваних дисковим інтерфейсом, або за допомогою додаткового кодування інформації, записаної на диск і використовується для виправлення випадкових збоїв. По суті, якщо контролер може визначити положення помилкового розряду, то для відновлення даних потрібен лише один біт парності. Зменшення кількості контрольних дисків до одного на групу знижує надмірність ємності до цілком розумних розмірів. Часто кількість дисків у групі дорівнює 5 (4 диска даних плюс 1 контрольний).

Реалізація полягає у наступному. Потік даних розбивається на блоки на рівні байт (хоча можливо і на рівні біт) і записується одночасно на всі диски масиву, окрім диска, який виділений для зберігання контрольних сум, що обчислюються при записі даних. Поломка будь-якого з дисків масиву не призведе до втрати інформації, яку можна відновити обчисленням операції "XOR", застосованою до інформації на дисках, що залишилися.

Цей рівень має набагато меншу надмірність, ніж RAID 2, в схемі якого більшість дисків, що зберігають контрольну інформацію, потрібні для визначення несправного розряду. Як правило, RAID-контролери можуть отримати дані про помилку за допомогою механізмів відстежування випадкових збоїв (за допомогою розшифровки сигналів від дисків або додаткового кодування). Завдяки розбиттю даних на блоки RAID 3 має високу продуктивність. При зчитуванні інформації не проводиться звернення до диска з контрольними сумами (в разі відсутності збою), що відбувається всякий раз при операції запису. Оскільки при кожній операції введення/виведення проводиться звернення практично до всіх дисків масиву, одночасна обробка декількох запитів неможлива. Цей рівень використовують для додатків з файлами великого об’єму і малою частотою звернень (в основному це сфера мультимедіа). Використання лише одного диска для зберігання

контрольної інформації пояснює той факт, що коефіцієнт використання дискового простору досить високий (і як наслідок цього – відносно низька вартість). Крім того, перевагою RAID 3 є незначне зниження продуктивності при збої і швидке відновлення інформації, недоліком – складність реалізації.

Переваги:

- дуже висока швидкість передачі даних;
- відмова диска мало впливає на швидкість роботи масиву;
- малі накладні витрати для реалізації надмірності.

Недоліки:

- складна реалізація;
- низька продуктивність при великій інтенсивності запитів даних невеликого об'єму.

RAID 4. Відмовостійкий масив незалежних дисків з диском парності Independent Data disks with shared Parity disk, що розділяється).

RAID рівня 4 підвищує продуктивність передачі невеликих обсягів даних за рахунок паралелізму, даючи можливість виконувати більше одного звернення введення/виведення до групи в одиницю часу. Логічні блоки передачі в даному випадку не розподіляються між окремими дисками, замість цього кожен індивідуальний блок потрапляє на окремий диск.

RAID рівня 4 багато в чому схожий з рівнем RAID 3. Потік даних розділяється не на рівні байтів, а на рівні блоків, кожен з яких записується на окремий диск. Після запису групи блоків обчислюється контрольна сума, яка записується на виділений для цього диск.

Завдяки більшому ніж в RAID 3 розміру блоку можливе одночасне виконання декількох операцій читання. RAID 4 підвищує продуктивність передачі файлів малого об'єму (за рахунок розпаралелювання операції зчитування). Але оскільки при записі повинна змінюватися контрольна сума на виділеному диску, одночасне виконання операцій неможливе (при наявності асиметричності операцій введення і виводу). Цей рівень має всі недоліки RAID 3 і не забезпечує переваги в швидкості при передачі даних великого об'єму. Схема зберігання розроблялася для застосувань, в яких дані спочатку розбиті на невеликі блоки, тому немає необхідності розбивати їх додатково. RAID 4 - непогане рішення для файл-серверів, інформація з яких в основному прочитується і рідко записується. Ця схема зберігання даних має невисоку вартість, але її реалізація досить складна, як і відновлення даних при збої.

Переваги:

- дуже висока швидкість передачі даних;
- відмова диска мало впливає на швидкість роботи масиву;

- малі накладні витрати для реалізації надмірності.

Недоліки:

- досить складна реалізація;
- дуже низька продуктивність при записі даних;
- складне відновлення даних.

RAID 5. Відмовостійкий масив незалежних дисків з розподіленою парністю (Independent Data disks with distributed parity blocks).

Відмовостійкий масив незалежних дисків з розподілом контрольних сум (масив з парністю, що обертається). Найпоширеніший рівень. Блоки даних і контрольні суми циклічно записуються на всі диски масиву, відсутній виділений диск для зберігання інформації про парність: немає асиметричності конфігурації дисків.

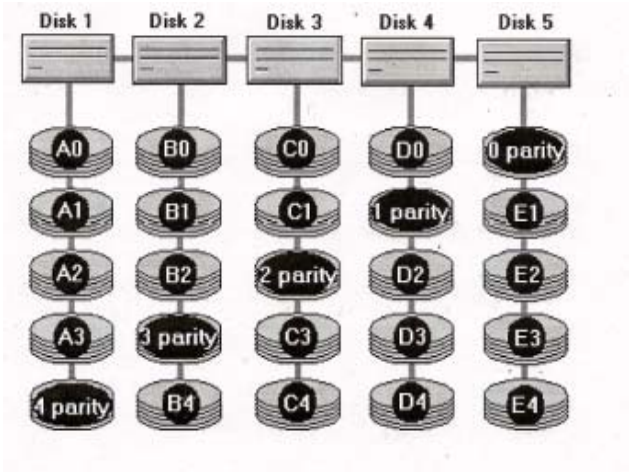


Рис.3.11. RAID рівня 5.

В RAID 5 всі диски масиву мають однаковий розмір, але один з них невидимий для операційної системи. Наприклад, якщо 3 диски мають розмір 1 Гб, то фактично розмір масиву складає 2 Гб, 1 Гб відводиться на контрольну інформацію. В разі додавання четвертого диска операційна система бачитиме 3 Гб, 1 Гб призначений для зберігання контрольних сум. Найбільший недолік рівнів RAID від 2-го до 4-го - це наявність окремого (фізичного) диска, що зберігає інформацію про парність. Операції зчитування не вимагають звернення до цього диска і, як наслідок, швидкість їх виконання досить висока, але при кожній операції запису на ньому змінюється інформація, тому схеми RAID 2-4 не дозволяють проводити паралельні операції запису. RAID 5 не має цього недоліку, оскільки контрольні суми записуються на всі диски масиву, що робить можливим виконання декількох операцій зчитування або запису одночасно. Грамотна реалізація цього рівня в разі масиву з N дисків дозволяє одночасно обробляти N/2 блоків даних. RAID 5 має досить високу швидкість запису- зчитування (швидкість читання нижча, ніж в RAID 4) і малу надмірність, тобто він економічніший.

Переваги:

- висока швидкість запису даних; досить висока швидкість читання даних;
- висока продуктивність при великій інтенсивності запитів зчитування/запису даних;

- малі накладні витрати для реалізації надмірності.

Недоліки:

- низька швидкість читання/запису даних малого об'єму при одиничних запитах;
- дуже низька продуктивність при записі даних;
- досить складна реалізація;
- складне відновлення даних.

RAID 6. Відмовостійкий масив незалежних дисків з двома незалежними розподіленими схемами парності (Independent Data disks with two independent distributed parity schemes).

Цей термін використовують як мінімум в трьох різних випадках. Як часто буває при бурхливому розвитку технології, різні виробники комп'ютерного устаткування розробляють свої власні розширення стандартної архітектури.

Деякі виробники беруть масив RAID 5, додають надлишкові джерела живлення і, можливо, диск "гарячого" резервування (вільний диск, на який автоматично переноситься інформація з накопичувача, що вийшов з ладу, тим самим відновлюється стан системи до збою) і називають цю конфігурацію RAID 6. Інші просто трохи змінюють процедуру запису, використовувану в RAID 5, і отримують RAID 6.

Але справжній RAID 6 - це відмовостійкий масив незалежних дисків з розподілом контрольних сум, обчислених двома незалежними способами. Цей рівень багато в чому схожий з RAID 5, але наявність двох незалежних схем контролю парності дозволяє зберігати працездатність системи при одночасному виході з ладу двох накопичувачів. Для обчислення контрольних сум в RAID 6 використовується алгоритм, побудований на основі коди Ріда-Саломона. Цей рівень має дуже високу відмовостійку, велику швидкість зчитування (дані зберігаються блоками, немає виділених дисків для зберігання контрольних сум), але через великий об'єм контрольної інформації - низьку швидкість записування. Він дуже складний в реалізації, характеризується низьким коефіцієнтом використання дискового простору (для масиву з п'яти дисків він складає всього 60%, але із зростанням числа дисків ситуація виправляється).

"Дійсний" RAID 6 по багатьом характеристикам програє іншим рівням, тому на сьогоднішній день не реалізований жодною фірмою, що проводить RAID-системи. Всі моделі RAID 6, які зустрічаються на ринку, це невеликі модифікації RAID 5.

Переваги:

- висока відмовостійкість;
- досить висока швидкість обробки запитів;

- відносно малі накладні витрати для реалізації надмірності.

Недоліки:

- низька швидкість читання/запису даних малого об'єму при одиничних запитах;
- дуже складна реалізація;
- складне відновлення даних;
- дуже низька швидкість запису даних.

RAID 7. Відмовостійкий масив, оптимізований для підвищення продуктивності (Optimized Asynchrony for High I/O Rates as well as High Data Transfer Rates)

RAID 7 є зареєстрованою торгівельною маркою корпорації Storage Computer. Багато в чому він схожий на RAID 4 з можливістю кешування даних. До складу RAID 7 входить контролер із вбудованим мікропроцесором під управлінням операційної системи реального часу (SOS). Вона дозволяє обробляти всі запити на передачу даних (як між окремими дисками, так і між масивом і комп’ютером) синхронно і незалежно. Блок обчислення контрольних сум інтегрований з блоком буферизації, для зберігання інформації про парність використовується окремий диск, який може бути розміщений на будь-якому каналі. RAID 7 має високу швидкість передачі даних і обробки запитів, добре масштабування (при збільшенні числа дисків підвищується швидкість запису). Найбільшим недоліком цього рівня є вартість його реалізації.

Переваги:

- дуже висока швидкість передачі даних і висока швидкість обробки запитів (у 1.5-6 разів вище за інші стандартні рівні RAID);
- висока масштабованість хост-интерфейсів (до 12-ти інтерфейсів і до 48-ми дисків; Для обчислення парності немає необхідності в додатковій передачі даних.

Недоліки:

- складна реалізація;
- дуже висока вартість на одиницю об'єму;
- не може обслуговуватися користувачем;
- потрібно використовувати блок безперебійного живлення для запобігання втраті даних з кеш-пам'яті (втім, UPS і так практично завжди застосовується для живлення серверів);
- короткий гарантійний термін.

RAID 10 (RAID 1+0). Відмовостійкий масив з дублюванням і паралельною обробкою

Комбінація рівнів 1 і 0. Кожен фізичний диск рівня RAID 0 замінюється масивом RAID 1. Це забезпечує високу передачу даних (сервер бачить масив як RAID 0) і високе їх

збереження, але значно обмежує масштабування, і коефіцієнт використання дискового простору виходить дуже низьким - всього 25%.

Переваги:

- дуже висока швидкість запису даних при не менш високій надійності.

Недоліки:

- дуже висока вартість;
- обмежене масштабування.

RAID 30 (RAID 3+0). Відмовостійкий масив з паралельною передачею даних і підвищеною продуктивністю.

Масив нульового рівня, роль дисків якого грають масиви RAID 3, поєднує продуктивність RAID 0 і відмовостійкість RAID 3. Оскільки в схемі RAID 3 дисковий простір використовується раціональніше, ніж в RAID 1, то в масиві RAID 3+0 коефіцієнт його використання вищий, ніж в RAID 1+0, і дорівнює 40%. Ця схема має обмежене масштабування.

Переваги:

- висока відмовостійкість;
- висока продуктивність.

Недоліки:

- велика вартість;
- обмежене масштабування.

RAID 50 (RAID 5+0). Відмовостійкий масив з розподіленою парністю і підвищеною продуктивністю.

Масив нульового рівня, роль дисків якого грають масиви RAID 5. Він об'єднує в собі відмовостійкість і високу продуктивність для застосувань з великою інтенсивністю запитів і високу швидкість передачі даних. RAID 5+0 володіє високою продуктивністю та вартістю. Ця схема теж має обмежене масштабування. Можливий ще варіант RAID 5+3, коли фізичні диски масиву RAID 5 замінюються масивами RAID 3.

Переваги:

- висока відмовостійкість і продуктивність;
- досить висока швидкість читання даних;
- висока продуктивність при великій інтенсивності запитів читання/запису даних;
- малі накладні витрати для реалізації надмірності.

Недоліки:

- велика вартість;
- обмежене масштабування.

4. Системи паралельної обробки

4.1. Класифікація систем паралельної обробки

Класифікація Фліна систем паралельної обробки.

Процес рішення довільної задачі можна представити як дію певної послідовності команд програми (потоку команд) на відповідну послідовність даних (потік даних), що викликаються цією послідовністю команд. Різні способи організації паралельної обробки інформації можна представити як способи організації одночасної дії одного або декількох потоків команд на один або декілька потоків даних.

Для такої класифікації корисно ввести поняття множинності потоків команд і даних - наявність в системі декількох послідовностей команд, що знаходяться у стадії реалізації, або декількох потоків даних, що піддаються обробці командами.

Виходячи з цього, всі системи можуть бути розбиті на чотири класи (Флінна 1966).

1) Системи з одиночним потоком команд і одиночним потоком даних (ОКОД, рис.4.1) - звичайні однопроцесорні ЕОМ, які включають один процесор, запам'ятовуючий пристрій для команд і даних та пристрій керування. У даних системах широко використовується перший спосіб організації паралельних обчислень, що поєднує в часі різні етапи вирішення завдань: введення, вивід і обробка інформації.

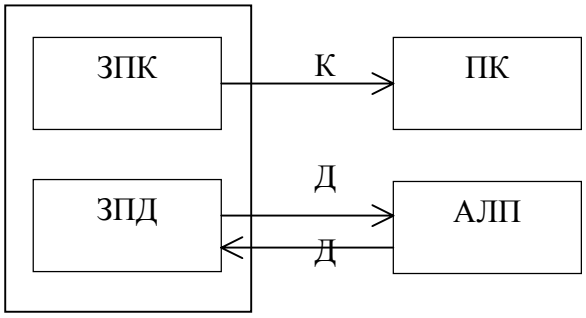


Рис. 4.1. Системы ОКОД.

2) Системи з множинним потоком команд і одиночним потоком даних (МКОД, рис.4.2). Проте не існує класу завдань, в якому одна і та ж послідовність даних оброблялася декількома різними програмами. Через це в чистому вигляді така схема до цих пір не реалізована. Проте, на практиці реалізована схема, в якій один потік команд розділяється на декілька потоків мікрооперацій, кожна з яких виконується спеціально налаштованим на неї пристроєм. Потік

даних проходить послідовно через всі спеціалізовані АЛП. Системи такого класу називаються конвеєрними (рис.4.2б).

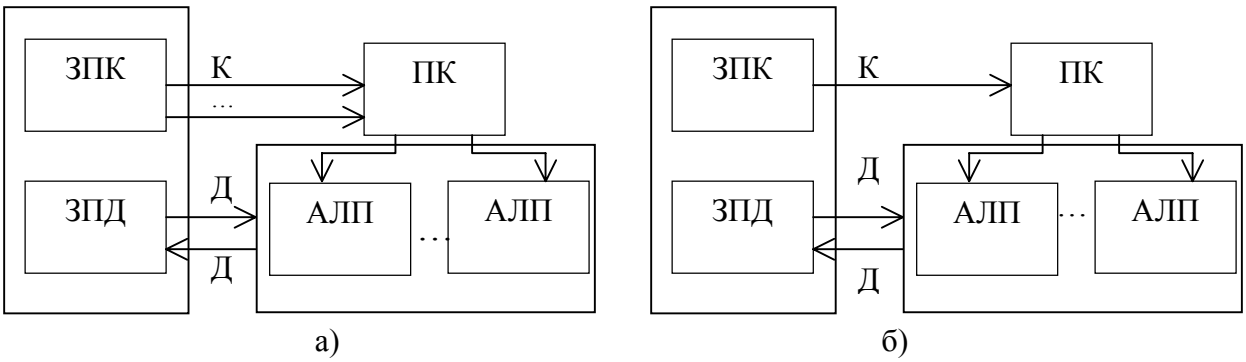


Рис. 4.2. Системы МКОД.

3) Системи з одиночним потоком команд і множинним потоком даних (ОКМД, рис 4.3). У даних системах за однією і тією ж програмою обробляється декілька потоків даних, кожен з яких обробляється своїм АЛП, які працюють під загальним управлінням.

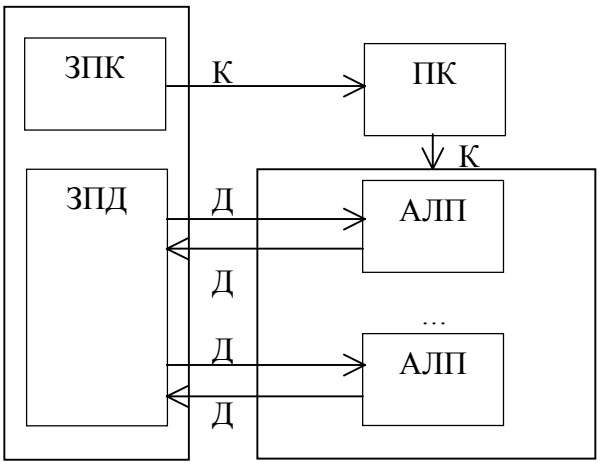


Рис. 4.3. Системи ОКМД.

4) Системи з множинним потоком команд і множинним потоком даних (МКМД). Можливі два способи побудови такого типа систем: у вигляді сукупності систем ОКОД або за схемою рис.4.4. У першому випадку для кожної послідовності команд і даних є власний ЗП, тоді як в другому випадку всі команди і дані розташовуються в загальних ЗП. Перший варіант (багатомашинний комплекс) пристосований для вирішення потоку незалежних завдань і при цьому досягається істотне підвищення продуктивності.

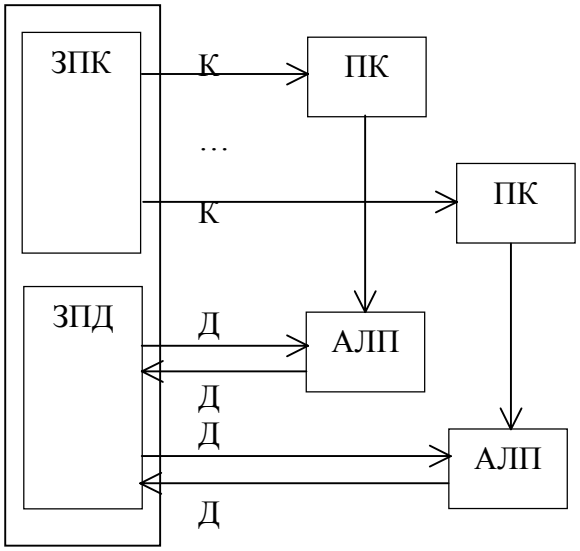


Рис. 4.4. Системи МКМД.

Класифікація Джонсона.

Е. Джонсон запропонував проводити класифікацію МКМД-архітектур на основі структури пам'яті та реалізації механізму взаємодії і синхронізації між процесорами.

За структурою оперативної пам'яті існуючі системи поділяються на дві великі групи: або це системи із загальною пам'яттю, що прямо адресується всіма процесорами, або це системи з розподіленою пам'яттю, кожна частина якої доступна лише одному процесору. Одночасно з цим і для міжпроцесорної взаємодії існують дві альтернативи - через змінні, що розділяються, або за допомогою механізму передачі повідомлень. Виходячи з таких припущень, можна отримати чотири класи МКМД-архітектур, що уточнюють класифікацію Фліна (табл.4.1).

Базуючись на цьому Джонсон вводить наступні найменування для деяких класів:

- обчислювальні системи, що використовують пам'ять, що розділяється та використовується для міжпроцесорної взаємодії і синхронізації – «системи з пам'яттю, що розділяється» (клас 1); приклад CRAY Y-MP;
- системи, в яких пам'ять розподілена по процесорах, а для взаємодії і синхронізації використовується механізм передачі повідомлень - «архітектура з передачею повідомлень» (клас 3); приклад NCube;
- системи з розподіленою пам'яттю і синхронізацією через змінні, що розділяються - «гібридна архітектура» (клас 2); приклад BBN Butterfly.

Класифікація Дункана заснована на декількох ознаках:

1) З класу паралельних машин мають бути виключені ті, в яких паралелізм закладений лише на найнижчому рівні, включаючи:

Табл 4.1. Класифікація Джонсона для систем МКМД за Фліном.

Обмін даними	Пам'ять	
	загальна	розподілена
загальні дані	GMSV - General Memory-Shared variables (загальна пам'ять та змінні, що розділяються) Клас 1. «Систем з пам'яттю, що розділяється»	DMSV-Distributed Memory, Shared variables (розподілена пам'ять та змінні, що розділяються) Клас 2. «Гібридна архітектура»
передача даних	GMMP - General Memory, Message propagation (загальна пам'ять та передача повідомлень)	DMMP - Distributed Memory, Message propagation) (розподілена пам'ять та передача повідомлень) Клас 3. «Архітектури з передачею повідомлень»

- конвеєризацію на етапі підготовки і виконання команди (instruction pipelining), тобто часткове перекриття таких етапів, як дешифрування команди, обчислення адрес операндів, вибірка операндів, виконання команди і збереження результату;
- наявність в архітектурі декількох функціональних пристроїв, що працюють незалежно, зокрема, можливість паралельного виконання логічних і арифметичних операцій;
- наявність окремих процесорів введення-виведення, що працюють незалежно і паралельно з основними процесорами.

Причини виключення перерахованих вище особливостей автор пояснює таким чином. Якщо розглядати комп'ютери, що використовують лише паралелізм низького рівня, нарівні зі всіма іншими, то, по-перше, практично всі існуючі системи будуть класифіковані як паралельні (що не є позитивним чинником для класифікації), і, по-друге, такі машини погано вписуватимуться в будь-яку модель або концепцію, що відображає паралелізм високого рівня.

2)Класифікація має бути погодженою з класифікацією Фліна, що показала правильність вибору ідеї потоків команд і даних.

3)Класифікація повинна описувати архітектури, які однозначно не вкладаються в систематику Фліна, але проте відносяться до паралельної архітектури (наприклад, векторно-конвеєрні).

Враховуючи вищевикладені вимоги, Дункан дає неформальне визначення паралельної архітектури, причому саме неформальність дала йому можливість включити в даний клас комп'ютери, які раніше не вписувалися в систематику Флінна.

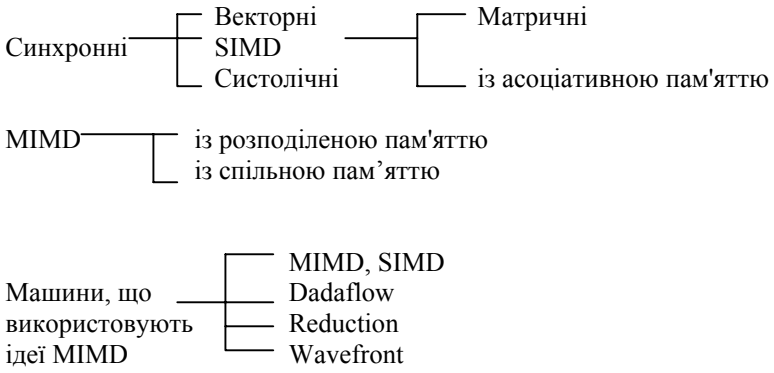


Рис.4.5. Класифікація Дункана

Паралельна архітектура - це такий спосіб організації обчислювальної системи, при якому допускається, аби множина процесорів (простих або складних) могла б працювати одночасно, взаємодіючи у міру потреби один з одним. Слідуючи цьому визначенню, всю різноманітність паралельних архітектур Дункан систематизує так, як це показано на рис.4.5.

По суті систематика дуже проста: процесори системи працюють або синхронно, або незалежно один від одного, або в архітектуру системи закладена інша модифікація ідеї МКМД. На наступному рівні відбувається деталізація цих трьох класів.

Систолічна архітектура (частіше називають систолічними масивами) є множиною процесорів, об'єднаних регулярним чином (наприклад, система WARP). Звернення до пам'яті можливо тільки через процесори на кордоні масиву. Вибірка операндів з пам'яті та передача даних по масиву здійснюється в одному і тому ж темпі. Напрям передачі даних між процесорами фіксований. Кожен процесор за інтервал часу виконує невелику інваріантну послідовність дій.

Гібридні МКМД/ОКМД-архітектури, обчислювальні системи dataflow, reduction і wavefront здійснюють паралельну обробку інформації на основі асинхронного управління, як і МКМД-системи. Але вони виділені в окрему групу, оскільки мають ряд специфічних особливостей, яких не має у систем, що традиційно відносяться до МКМД.

МКМД/ОКМД - типово гібридна архітектура. Вона передбачає, що в МКМД-системі можна виділити групу процесорів, що є підсистемою, та яка працює в режимі ОКМД (наприклад, PASM, Non-Von). Такі системи відрізняються відносною гнучкістю, оскільки допускають реконфигурацію відповідно до особливостей прикладного завдання, що вирішується.

Останні три види архітектури використовують нетрадиційні моделі обчислень. *Dataflow-машини* використовують модель, в якій команда може виконуватися відразу ж, як тільки

обчислені необхідні операнди. Таким чином, послідовність виконання команд визначається залежністю за даними, яка може бути виражена, наприклад, у формі графа.

Інша модель обчислень використовується в reduction-машинах: команда стає доступною для виконання тоді і тільки тоді, коли результат її роботи потрібен іншій доступній для виконання команді в якості операнда

Архітектура *wavefront array* об'єднує в собі ідею систолічної обробки даних та модель обчислень, що використовується в dataflow-машинах. У даній архітектурі процесори об'єднуються в модулі; зв'язки, по яких процесори можуть взаємодіяти один з одним, фіксуються. Проте, в протилежність ритмічній роботі систолічних масивів, дана архітектура використовує асинхронний механізм зв'язку з підтвердженням (handshaking), через це «фронт хвилі» обчислень може міняти свою форму під час руху по всій множині процесорів.

Класифікація Скіллікорна.

Пропонується розглядати архітектуру будь-якого комп'ютера як абстрактну структуру, що складається з чотирьох компонентів:

- процесор команд (IP - Instruction Processor) - функціональний пристрій, що працює як інтерпретатор команд (у системі, взагалі кажучи, може бути відсутнім);
- процесор даних (DP - Data Processor) - функціональний пристрій, що працює як перетворювач даних, відповідно до арифметичних операцій;
- ієрархія пам'яті (IM - Instruction Memory, DM - Data Memory) – запам’ятовуючий пристрій, в якому зберігаються дані і команди, що пересилаються між процесорами;
- перемикач - абстрактний пристрій, що забезпечує зв'язок між процесорами і пам'яттю.

Функції процесора команд багато в чому схожі з функціями пристроїв управління послідовних машин і, згідно з Д. Скіллікорном, зводяться до наступних:

- на основі свого стану і отриманою від DP інформації IP визначає адресу команди, яка виконуватиметься наступною;
- здійснює доступ до IM для вибірки команди;
- отримує і декодує вибрану команду;
- повідомляє DP команду, яку треба виконати;
- визначає адреси операндів і посилає їх в DP;
- отримує від DP інформацію про результат виконання команди.

Функції процесора даних роблять його багато в чому схожим на арифметичний пристрій традиційних процесорів:

- DP отримує від IP команду, яку треба виконати;
- отримує від IP адреси операндів;

- вибирає операнди з DM;
- виконує команду;
- запам'ятовує результат в DM;
- повертає в IP інформацію про стан після виконання команди.

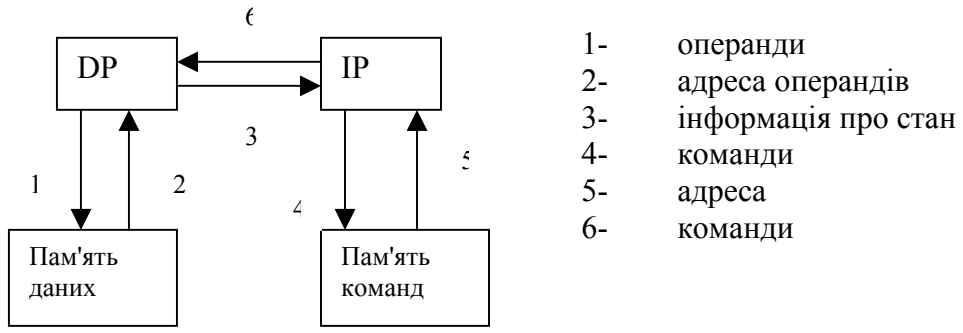


Рис. 4.6. Представлення фон-Неймоновской архітектури за Скіллікорном.

В термінах таким чином визначених основних частин комп'ютера структуру традиційної фон-нейманівської архітектури можна представити в наступному вигляді (рис.4.6).

Це один з найпростіших видів архітектури, що не містить перемикачів. Для опису паралельних обчислювальних систем автор фіксує чотири типи перемикачів, без якого-небудь явного зв'язку з типом пристроїв, які вони сполучають:

- - перемикач такого типа зв'язує пару функціональних пристроїв;
- *n-n* перемикач зв'язує і-й пристрій з однієї множини пристроїв з і-м пристроєм з іншої множини, тобто фіксує попарний зв'язок;
- *1-n* - перемикач сполучає один виділений пристрій зі всіма функціональними пристроями з деякого набору;
- *n x n* - кожен функціональний пристрій однієї множини може бути пов'язаний з будь-яким пристроєм іншої множини і навпаки.

Прикладів подібних перемикачів можна привести дуже багато. Так, всі матричні процесори мають перемикач типа 1-*n* для зв'язку єдиного процесора команд зі всіма процесорами даних. У комп'ютерах сімейства Connection Machine кожен процесор даних має свою локальну пам'ять, отже, зв'язок описуватиметься як *n-n*. В той самий час кожен процесор команд може зв'язатися з будь-яким іншим процесором, тому даний зв'язок буде описаний як *n* x *n*.

Класифікація Д. Скіллікорна складається з двох рівнів. На першому рівні вона проводиться на основі восьми характеристик:

- кількість процесорів команд (IP);
- кількість запам’ятовуючих пристроїв (модулів пам'яті) команд (IM);
- типа перемикача між IP і IM;
- кількість процесорів даних (DP);
- кількість запам’ятовуючих пристроїв (модулів пам'яті) даних (DM);
- тип перемикача між DP і DM;
- типа перемикача між IP і DP;
- типа перемикача між DP і DP.

Використовуючи введені характеристики і передбачаючи, що розгляд кількісних характеристик можна обмежити лише трьома можливими варіантами значень: 0, 1 і *n* (тобто більше одного) можна отримати 28 класів архітектури.

У класах 1-5 знаходяться комп'ютери типа dataflow і reduction, що не мають процесорів команд в звичайному розумінні цього слова.

Клас 6 — це класична фон-нейманівська послідовна машина.

Всі різновиди матричних процесорів містяться у класах 7-10.

Класи 11 і 12 відповідають комп'ютерам типа МКОД класифікації Фліна і на теперішній час, скоріш за все, не заповнені.

Класи з 13-го по 28-й займають всілякі варіанти мультипроцесорів, причому в 13-20 класах знаходяться машини з досить звичною архітектурою, в той час, як архітектура класів 21-28 виглядає екзотично.

4.2 Матричні системи

Найбільш поширеними системами класу ОКМД є матричні системи, які краще всього пристосовані для вирішення завдань, що характеризуються паралелізмом незалежних об'єктів, або паралелізмом даних. Організація таких систем має наступний вигляд: загальний керуючий пристрій, що генерує потік команд, та велике число пристроїв, що працюють паралельно та обробляють кожен свій потік даних. Отже, продуктивність системи такого вигляду дорівнює сумі продуктивності всіх оброблювальних пристроїв. На практиці для підвищення ефективності необхідно організувати зв'язки між оброблювальними пристроями. Характер цих зв'язків ш визначає властивості систем.

Класичним прикладом багатопроцесорної системи є система ILLIAC IV (рис.4.7), основні ідеї якої були запозичені з системи SOLOMON. За первинним проектом система

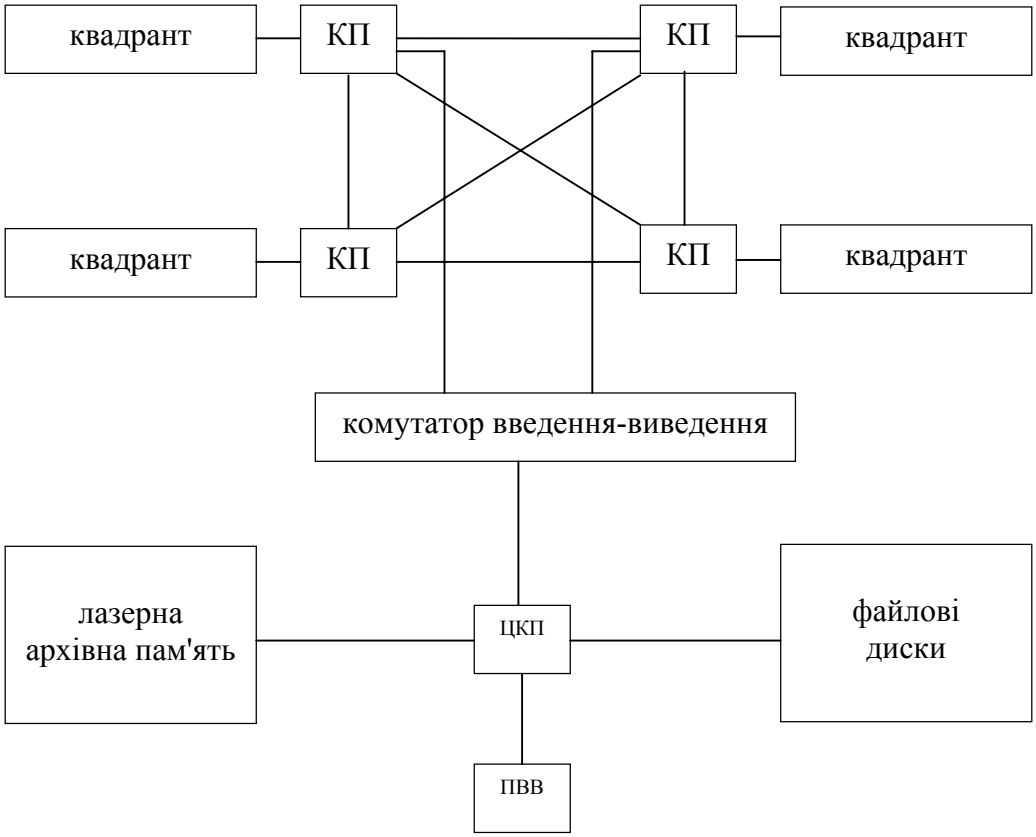


Рис. 4.7. Система ILLIAC IV.

ILLIAC IV повинна була включати 256 процесорних елементів (ПЕ), розбитих на 4 групи, - квадранти, кожен з яких повинен управлятися спеціальним процесором (УП). Управління всією системою, що містить окрім УП і ПЕ також зовнішню пам'ять і устаткування введення-виведення, передбачалося від центрального керуючого процесора.

У кожному квадранті міститься 64 ПЕ утворюють матрицю розміром 8*8, причому зв'язок із зовнішнім середовищем мають всі ПЕ без виключення. Кожен ПЕ складається з власне процесора і ОЗП. Продуктивність кожного ПЕ близько 3 млн. операцій в секунду. Ємність ОЗП кожного елементу 2048 64-розрядних слів. Кожен процесор має лічильник адрес і індексний регістр і відрізняється від попередніх розробок гнучкістю у формуванні адрес операндів.

Надвисока продуктивність системи досягається лише на певних типах завдань: операції над матрицями, швидке перетворення Фур'є, лінійне програмування, обробка сигналів, тобто на завданнях, де має місце паралелізм даних або паралелізм незалежних об'єктів. Для розробки програмного забезпечення таких систем необхідне створення спеціалізованих мов програмування або компіляторів.

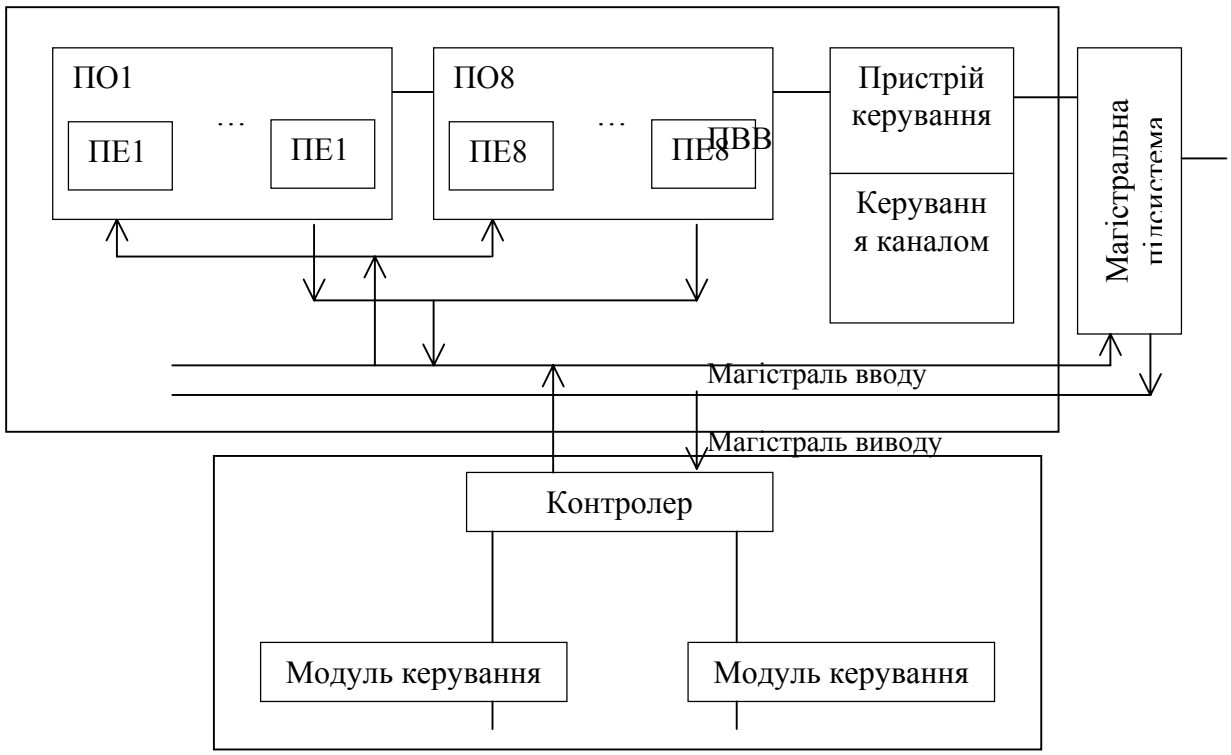


Рис.4.8. Система ПС2000.

Іншим вдалим прикладом є система ПС 2000 («Паралельна система 2000», рис.4.8), яка орієнтована на вирішення завдань, що характеризуються паралелізмом даних, незалежних гілок та об'єктів.

Центральна частина системи - мультипроцесор ПС 2000, що складається з вирішального поля і пристрою керування мультипроцесором. Вирішальне поле будується з одного, двох, чотирьох або восьми пристроїв обробки (ПО), в кожному з яких 8 процесорних елементів. Ємність оперативної пам'яті ПЕ - 4096 або 16384 24-розрядних слова.

Пристрій керування містить блок мікропрограмного управління ємністю 16384 мікрокоманди, ОЗУ ємкістю 4096 або 16384 24-розрядних слів і АЛП. У комплект постачання входить набір мікропрограм базової операційної системи, орієнтованої на обробку матриць, реалізацію швидкого перетворення Фур'є і вирішення завдань математичної статистики, спектрального аналізу, лінійного і динамічного програмування.

Обчислювальний процес в системі ПС 2000 складається з трьох етапів: процесу в моніторинговій керуючій системі, виконання програми і введення/виведення даних. Основним є процес в моніторинговій системі, який ініціює та синхронізує інші процеси. Під його керівництвом завантажується набір мікропрограм і програма, які в процесі обробки даних можуть змінюватися.

Система ПС 2000 володіє перед системою ПХ1АС IV рядом переваг:

- процесорні елементи мають великі можливості, що визначається наявністю надоперативної регістрової пам'яті, наявністю процесорів для операцій над адресами і процесора активації;
- завдяки наявності власної пам'яті і індексної арифметики, використовуваної для організації лічильників адрес, можливе поєднання обміну інформацією між модулями пам'яті ПЕ, АЛП і пристроями введення-виведення;
- велика ємність пам'яті ПЕ і багаторівнева система переривань дозволяють організацію мультипрограмного режиму з виділенням незалежних ресурсів для кожного завдання;
- дворівневе управління (командне і мікрокомандне) забезпечує більш ефективне програмування;
- можливе нарощування системи модулями по 8 ПЕ без зміни засобів управління;
- невисока вартість.

4.3 Асоціативні системи

До числа систем класу ОКМД відносяться асоціативні системи. Вони, як і матричні системи, характеризуються великою кількістю операційних пристроїв, здатних одночасно за командами одного керуючого пристрою вести обробку декількох потоків даних. Проте вони істотно відрізняються способом формуванням потоку даних. У матричних системах дані надходять на обробку від загальних або роздільних запам'ятовуючих пристроїв з адресною вибіркою інформації, або безпосередньо від пристроїв - джерел даних. У асоціативних системах інформація на обробку надходить від асоціативних запам'ятовуючих пристроїв (АЗП), що характеризуються тим, що інформація з них вибирається не за певною адресою, а за її змістом. Швидкість вибірки не залежить від кількості елементів пам'яті, а тільки від кількості розрядів ознаки. Для поліпшення властивостей систем часто АЗП будується таким чином, що крім асоціативної допускається і звичайна адресація, що представляється зручним при роботі з периферійними пристроями. Також в АЗП реалізовані функції, що дозволяють здійснювати пошук не тільки за співпаданням значень комірок, а й за іншими умовами: вміст комірки більше (менше) ознаки, а також більше або дорівнює (менше або дорівнює).

Зазначені властивості дозволяють ефективно вирішувати завдання з наступних областей: обробка радіолокаційної інформації, розпізнавання образів, обробка різних знімків та іншої

інформації з матричною структурою даних. Причому, програмування таких завдань для асоціативних систем набагато простіше, ніж для традиційних.

Найбільш характерним представником групи асоціативних обчислювальних систем є система STARAN (США). Відмінність від описаних матричних систем полягає в наступному:

- асоціативна пам'ять є пам'яттю з багатовимірним доступом, тобто в неї можна звернутися як порозрядно, так і послівно;
- операційні процесорні елементи передбачені для кожного слова пам'яті;
- є унікальна схема перестановок для перегрупування даних у пам'яті.

Основним елементом системи є багатовимірна асоціативна матриця - асоціативний модуль (АМ), який являє собою квадрат з 256 розрядів на 256 слів (Рис.4.9), тобто містить в загальній складності 65535 біт даних. Для обробки даних є 256 ПЕ, які послідовно розряд за розрядом обробляють слова. Всі ПЕ працюють одночасно, по одній команді, котра видається пристроєм керування. Таким чином, відразу по одній команді обробляються всі вибрані за певними ознаками з пам'яті слова.

Базова конфігурація системи STARAN містить один асоціативний модуль (АМ). Проте їхня кількість може збільшуватися до 32-х. При цьому максимальний обсяг одночасно оброблюваної інформації досягає 256 КБайт.

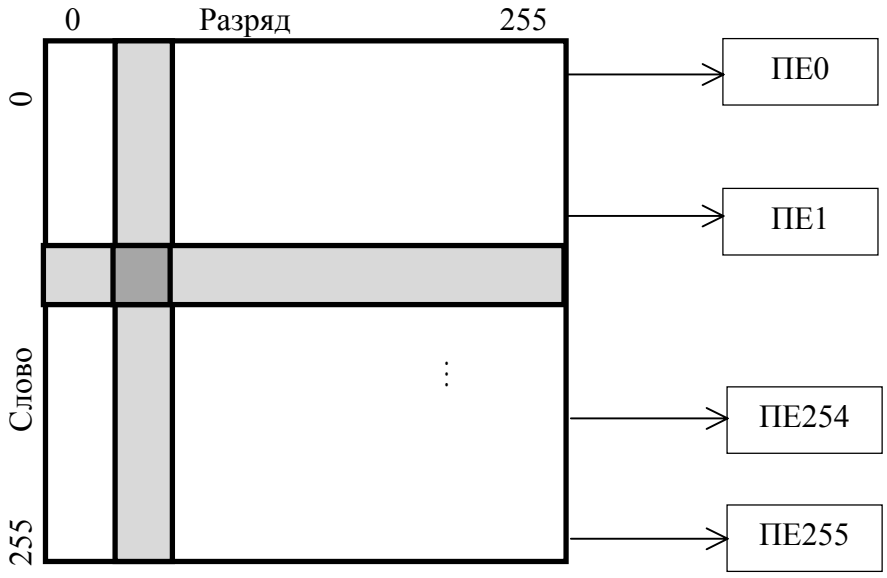


Рис.4.9. Процесорна обробка в системі STARAN.

Підсистема введення/виведення забезпечує підключення до системи периферійних пристроїв за чотирма типами інтерфейсів:

- інтерфейс прямого доступу для швидкого доступу та для нарощування пам'яті;

- буферізоване введення/виведення для зв'язку зі стандартними периферійними пристроями з обміном блоками даних;
- паралельне введення/виведення, що включає до 256 входів і виходів для кожної матриці та дозволяє збільшити швидкість обміну даними між матрицями, забезпечити зв'язок системи з високошвидкісними засобами введення/виведення і безпосередній зв'язок будь-якого пристрою з асоціативними пристроями.

Сукупність усіх зазначених особливостей дозволяє системі STARAN виконувати одночасно сотні й тисячі однакових операцій при вирішенні певного класу задач.

4.4 Однорідні системи та середовища

У системах паралельної обробки, що розглядаються вище, не ставиться ніяких умов або обмежень щодо складу та функцій обробляючих пристроїв, а також зв'язків між ними.

На початку 60-х років Е.В. Євреїнов і Ю.Г. Косарєв запропонували інший підхід до побудови систем, в основі якого лежать принципи:

- паралельність операцій;
- змінність логічної структури;
- конструктивна однорідність елементів і зв'язків між ними.

Перший принцип базується на *аксіомі паралельності завдань та алгоритмі*в: будь-яка складна задача може бути представлена у вигляді пов'язаних між собою простих підзадач і для будь-якої складної задачі може бути запропоновано паралельний алгоритм, що допускає її ефективне рішення.

Другий принцип базується на *аксіомі змінності логічної структури*: процес вирішення складної задачі може бути представлений певною структурною моделлю, що включає в себе підзадачі та зв'язки між ними.

Третій принцип базується на *аксіомі конструктивної однорідності елементів і зв'язків*: всі прості завдання отримуються шляхом поділу складної задачі на частини, а тому всі ці прості завдання приблизно однакові за об'ємом обчислень і пов'язані між собою однаковими схемами обміну. Це означає, що система для вирішення складного завдання може бути побудована з однакових оброблюючих елементів, пов'язаних між собою однаковим чином.

У розглянутих вище комплексах і системах в тій чи іншій мірі використовуються зазначені вище три принципи. Однорідні системи повністю використовують всі три принципи.

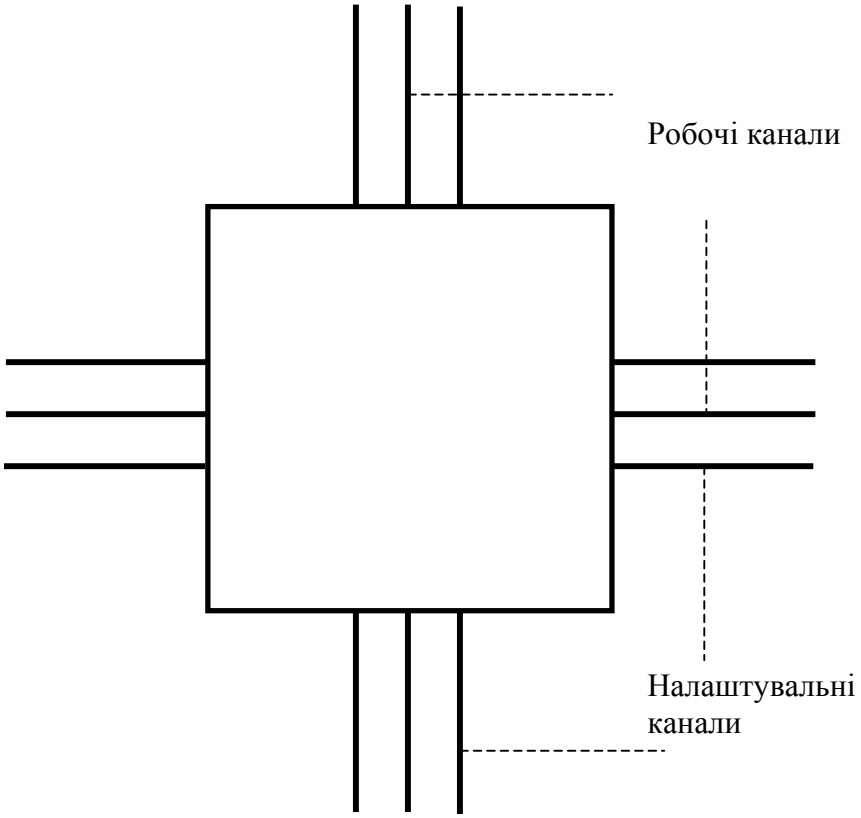


Рис.4.10. Елемент однорідного обчислювального середовища.

Важливим аспектом роботи однорідної системи є програмування її структури - налаштування, яке здійснюється за допомогою спеціальних регістрів налаштування, що входять до складу модулів міжмашинного зв'язку.

Розвитком однорідних систем є однорідні обчислювальні середовища, які являють собою в загальному випадку n-мірну решітчасту структуру. В окремому випадку це двомірна структура з квадратних клітин, що заповнюють площину. Кожна клітина (рис.4.10) у двомірній структурі з'єднується з чотирма сусідніми.

Між двома сусідніми клітинами проходять два канали налаштування і один канал передачі робочої інформації. Кожен елемент однорідного обчислювального середовища складається з комутаційних і функціональних компонентів. Функціональний компонент реалізує повноцінну систему логічних функцій (наприклад, АБО-НІ або І-НЕ). Комутаційні й функціональні компоненти дозволяють шляхом відповідного налаштування при достатньому їхньому числі реалізовувати будь-яку структуру.

5. Конвеєрні системи

Конвеєризація (або конвеєрна обробка) в загальному випадку полягає у розподілі виконуваної функції на більш дрібні частини, які називаються ступенями, і виділенні для кожної з них окремого блоку апаратури. Так обробку будь-якої машинної команди можна розділити на кілька етапів (ступенів), організувавши передачу даних від одного етапу до наступного. При цьому конвеєрну обробку можна використовувати для поєднання етапів виконання різних команд в часі. Продуктивність при цьому зростає завдяки тому, що одночасно на різних етапах конвеєра виконуються декілька команд. Конвеєрна обробка такого роду широко застосовується у всіх сучасних швидкодіючих процесорах.

5.1 Приклад організації найпростішого конвеєру

Для ілюстрації основних принципів побудови процесорів ми будемо використовувати найпростішу архітектуру, яка містить цілочисельні регістри загального призначення, регістри з плаваючою точкою і лічильник команд. Будемо вважати, що набір команд нашого процесора включає типові арифметичні і логічні операції, операції з плаваючою точкою, операції пересилання даних, операції керування потоком команд і системні операції. В арифметичних командах використовується триадресний формат, типовий для RISC-процесорів, а для звернення до пам'яті використовуються операції завантаження і запису вмісту регістрів в пам’ять.

Виконання типової команди розділимо на наступні етапи:

- вибирання команди - IF (за адресою, що задається лічильником команд, з пам'яті обирається команда);
- декодування команди / вибірка операндів із регістрів - ID;
- виконання операції / обчислення ефективного адреси пам'яті - EX;
- звернення до пам'яті - MEM;
- запам'ятовування результату - WB.

Роботу конвеєра можна умовно представити у вигляді зміщених у часі етапів (табл.5.1). Цей рисунок добре відображає суміщення в часі виконання різних етапів команд.

Конвеєрізация збільшує пропускну здатність процесора (кількість команд, що завершуються в одиницю часу), але вона не скорочує час виконання окремої команди. У дійсності, вона навіть трохи збільшує час виконання кожної команди через накладні витрати,

Табл. 5.1. Робота найпростішого конвеєру.

Номер команди	Номер такта								
	1	2	3	4	5	6	7	8	9
Команда i	IF	ID	EX	MEM	WB				
Команда i+1		IF	ID	EX	MEM	WB			
Команда i+2			IF	ID	EX	MEM	WB		
Команда i+3				IF	ID	EX	MEM	WB	
Команда i+4					IF	ID	EX	MEM	WB

пов'язані з управлінням. Однак збільшення пропускної спроможності означає, що програма буде виконуватися швидше в порівнянні з простою неконвеєрною схемою.

Той факт, що час виконання кожної команди в конвеєрі не зменшується, накладає деякі обмеження на практичну довжину конвеєра:

- частота синхронізації не може бути вище, а, отже, такт синхронізації не може бути менше, ніж час, необхідний для роботи найбільш повільної ступені конвеєра;
- накладні витрати на організацію конвеєра виникають через затримку сигналів у конвеєрних регістрах та через перекоси сигналів синхронізації;
- конвеєрні регістри до тривалості такту додають час установки і затримку поширення сигналів.

У граничному випадку тривалість такту можна зменшити до суми накладних витрат і перекосу сигналів синхронізації, однак при цьому в такті не залишиться часу для виконання корисної роботи по перетворенню інформації.

Як приклад розглянемо неконвеєрну машину з п'ятьма етапами виконання операцій, які мають тривалість 50, 50, 60, 50 і 50 нс відповідно. Нехай накладні витрати на організацію конвеєрної обробки становлять 5 нс. Тоді середній час виконання команди в неконвеєрній машині дорівнюватиме 260 нс. Якщо використовується конвеєрна організація, тривалість такту буде дорівнювати тривалості найповільнішого етапу обробки плюс накладні витрати, тобто 65 нс. Цей час відповідає середньому часу виконання команди в конвеєрі. Таким чином, прискорення, що отримане в результаті конвеєризації, дорівнюватиме:

$$\frac{час\_виконання\_команди\_в\_неконвеєрному\_режимі}{час\_виконання\_команди\_в\_конвеєрному\_режимі} = \frac{260}{65} = 4.$$

Конвеєризація ефективна тільки тоді, коли завантаження конвеєра близьке до повного, а швидкість подачі нових команд і операндів відповідає максимальній продуктивності конвеєра. Якщо відбудеться затримка, то паралельно буде виконуватися менше операцій і сумарна продуктивність знизиться. Такі затримки можуть виникати в результаті виникнення конфліктних ситуацій. У наступних розділах будуть розглянуті різні типи конфліктів, що виникають при виконанні команд в конвеєрі, і способи їх розв'язання.

5.2. Структурні конфлікти та способи їх мінімізації

Якщо будь-яка комбінація команд не може бути прийнята через конфлікт з ресурсів, то говорять, що в машині є структурний конфлікт. Основні причини конфліктів:

- машини містять не повністю конвеєрні функціональні пристрої, час роботи яких складає кілька тактів синхронізації конвеєра. У цьому випадку послідовні команди, які використовують даний функціональний пристрій, не можуть вступати до нього в кожному такті;
- недостатнє дублювання деяких ресурсів, що перешкоджає виконанню довільної послідовності команд в конвеєрі без його призупинення. Наприклад, машина може мати тільки один порт запису в регістровий файл, але при певних обставинах конвеєру може знадобитися виконати два записи в регістровий файл в одному такті.

Структурні конфлікти виникають, наприклад, в машинах, в яких є єдиний конвеєр пам'яті для команд і даних (табл. 5.2).

У цьому випадку, коли одна команда містить звернення до пам'яті за даними, воно буде конфліктувати з вибіркою більш пізньої команди з пам'яті. Щоб вирішити цю ситуацію,

Табл. 5.2. . Конфлікт на конвеєрі при зверненні до пам'яті.

Номер команди	Номер такту								
	1	2	3	4	5	6	7	8	9
Команда i	IF	ID	EX	MEM	WB				
Команда i+1		IF	ID	EX	MEM	WB			
Команда i+2			IF	ID	EX	MEM	WB		
Команда i+3				stall	IF	ID	EX	MEM	WB
Команда i+4						IF	ID	EX	MEM
Команда i+5							IF	ID	EX

можна просто призупинити конвеєр на один такт, коли відбувається звернення до пам'яті за даними. Подібне призупинення часто називаються «конвеєрною булькою» або просто булькою, оскільки булька проходить по конвеєру, займаючи місце, але не виконуючи ніякої корисної роботи.

Виникає питання: чому розробники допускають наявність структурних конфліктів? Для цього є дві причини: зниження вартості та зменшення затримки пристрою. Конвеєризація всіх функціональних пристроїв може виявитися занадто дорогою. Машини, що допускають два звернення до пам'яті в одному такті, повинні мати подвоєну пропускну спроможність шини пам'яті, наприклад, шляхом організації роздільних кешей для команд і даних. Аналогічно, повністю конвеєрний пристрій ділення з плаваючою точкою вимагає величезної кількості вентилів. Якщо структурні конфлікти не будуть виникати дуже часто, то може бути і не варто платити за те, щоб їх обійти. Як правило, можна розробити неконвеєрний, або не повністю конвеєрний пристрій, що має меншу загальну затримку, ніж повністю конвеєрний.

5.3. Конфлікти за даними, зупинки конвеєру і реалізація механізму обходів

Конфлікти за даними виникають в тому випадку, коли застосування конвеєрної обробки може змінити порядок звернень до операндів так, що цей порядок буде відрізнятися від порядку, який спостерігається при послідовному виконанні команд на не конвеєрній машині.

Розглянемо конвеєрне виконання послідовності команд в табл.5.3.

Табл. 5.3. Приклад конфлікту за даними.

ADD	R1, R2, R3	IF	ID	EX	MEM	WB				
SUB	R4, R1, R5		IF	ID	EX	MEM	WB			
ADD	R6, R1, R7			IF	ID	EX	MEM	WB		
OR	R8, R1, R9				IF	ID	EX	MEM	WB	
XOR	R10, R1, R11					IF	ID	EX	MEM	WB

У цьому прикладі всі команди, які йдуть за командою ADD, використовують результат її виконання. Команда ADD записує результат в регістр R1, а команда SUB читає це значення. Якщо не вжити жодних заходів для того, щоб запобігти цьому конфлікту, команда SUB прочитає неправильне значення і спробує його використовувати. Насправді значення, яке використовується командою SUB, є навіть невизначеним: хоча логічно припустити, що SUB

завжди буде використовувати значення R1, яке було присвоєно будь-якою командою, яка передувала ADD. Але це не завжди так. Якщо відбудеться переривання між командами ADD і SUB, то команда ADD завершиться, і значення R1 у цій точці буде відповідати результату ADD. Така непрогнозована поведінка є неприйнятною.

Проблема, поставлена в цьому прикладі, може бути розв'язана за допомогою досить простої апаратної техніки, яка називається пересиланням чи просуванням даних (data forwarding), Апаратура працює наступним чином. Результат операції АЛП з його вихідного регістра завжди знову подається назад на входи АЛП. Якщо апаратура виявляє, що попередня операція АЛП записує результат в регістр, що відповідає джерелу операнда для наступної операції АЛП, то логічні схеми управління вибирають у якості вхідного значення для АЛП результат, що надходить ланцюгом «обходу», а не значення, прочитане з регістру.

Класифікація конфліктів за даними.

Відомі три можливі типи конфліктів за даними в залежності від порядку операцій читання і запису. Розглянемо дві команди i та j, при цьому i передує j. Можливі наступні конфлікти:

- RAW (читання після запису) - j намагається прочитати операнд-джерело даних перш, ніж і туди їх запише. Таким чином, j може некоректно отримати старе значення. Це найбільш загальний тип конфліктів, спосіб їх подолання за допомогою механізму «обходів» розглянуто раніше;
- WAR (запис після читання) - j намагається записати результат в приймач перш, ніж він зчитується звідти командою i, так що і може некоректно отримати нове значення . Цей тип конфліктів як правило не виникає в системах із централізованим керуванням потоком команд, які забезпечують виконання команд в порядку їх надходження, оскільки подальший запис завжди виконується пізніше, ніж попереднє зчитування. Особливо часто конфлікти такого роду можуть виникати в системах, що допускають виконання команд не в порядку їх розташування в програмному коді;
- WAW (запис після запису) - j намагається записати операнд перш, ніж буде записаний результат команди i, тобто записи закінчуються в неналежному порядку, залишаючи в приймачі значення, записане командою i, а не j. Цей тип конфліктів присутній тільки в конвеєрах, які виконують запис з багатьох ступенів (або дозволяють команді виконуватися навіть у випадку, коли виконання попередньої команди призупинено).

Багато типів припинень конвеєра можуть відбуватися досить часто. Наприклад, для оператора A = B + C компілятор, швидше за все, згенерує наступну послідовність команд (табл .5.4).

Табл. 5.4. Виконання команди складання в конвеєрі.

LW R1, W	IF	ID	EX	MEM	WB				
LW R2, C		IF	ID	EX	MEM	WB			
ADD R3, R1, R2			IF	ID	stall	EX	MEM	WB	
SW A, R3				IF	stall	ID	EX	MEM	WB

Вочевидь, виконання команди ADD повинно бути призупинено до тих пір, поки не стане доступним операнд C, що надходить з пам’яті. Додаткової затримки виконання команди SW не станеться в разі застосування ланцюгів обходу для пересилання результату операції АЛП безпосередньо в регістр даних пам'яті для подальшого запису.

Для даного простого прикладу компілятор ніяк не може поліпшити ситуацію, однак у ряді більш загальних випадків він може реорганізувати послідовність команд так, щоб уникнути припинень конвеєра. Ця техніка називається плануванням завантаження конвеєра (планування потоку).

Нехай, наприклад, є послідовність операторів:

a = b + c; d = e – f.

Як згенерувати код, що не викликає зупинок конвеєра? Передбачається, що затримка завантаження з пам'яті складає один такт (табл.5.5).

Табл. 5.5. Оптимізація коду компілятором.

Неоптимізована послідовність команд	Оптимізована послідовність команд
LW R_b,b	LW R_b,b
LW R_c,c	LW R_c,c
ADD R_a,R_b,R_c	LW R_e,e
SW a,R_a	ADD R_a,R_b,R_c
LW R_e,e	LW R_f,f
LW R_f,f	SW a,R_a
SUB R_d,R_e,R_f	SUB R_d,R_e,R_f
SW d,R_d	SW d,R_d

У результаті усунені обидва блокування (командою LW Rc, с команди ADD Ra, Rb, Rc та командою LW Rf, f команди SUB Rd, Re, Rf).

У загальному випадку планування конвеєра може вимагати збільшення кількості регістрів. Таке збільшення може виявитися особливо суттєвим для машин, які можуть видавати на виконання декілька команд в одному такті.

Багато сучасних компіляторів використовують техніку планування команд для поліпшення продуктивності конвеєра. У найпростішому алгоритмі компілятор просто планує розподіл команд в одному і тому ж базовому блоці. Базовий блок представляє собою лінійну ділянку послідовності програмного коду, в якій відсутні команди переходу, за винятком початку і кінця ділянки (переходи всередину цієї ділянки теж повинні бути відсутніми). Планування такої послідовності команд здійснюється досить просто, оскільки компілятор знає, що кожна команда в блоці буде виконуватися, якщо виконується перша з них, і можна просто побудувати граф залежностей цих команд та спланувати їх так, щоб мінімізувати припинення конвеєра. Для простих конвеєрів стратегія планування на основі базових блоків цілком задовільна. Проте, коли конвеєризація стає більш інтенсивною і дійсні затримки конвеєра ростуть, потрібні більш складні алгоритми планування.

Існують апаратні методи, що дозволяють змінити порядок виконання команд програми так, щоб мінімізувати припинення конвеєра. Ці методи отримали загальну назву методів динамічної оптимізації (в англомовній літературі останнім часом часто застосовуються також терміни "out-of-order execution" - невпорядковане виконання та "out-of-order issue" - неупорядкована видача). Основними засобами динамічної оптимізації є наступні.

Розміщення схеми виявлення конфліктів в можливо більш низькій точці конвеєра команд так, щоб дозволити команді просуватися по конвеєру до тих пір, доки їй реально не буде потрібен операнд, що є результатом логічно більш ранньої, але ще не завершеної команди.

Альтернативним підходом є централізоване виявлення конфліктів на одній з ранніх ступенях конвеєра.

Буферизація команд, які очікують вирішення конфлікту, і видача наступних, логічно не пов'язаних команд, в «обхід» буфера. У цьому випадку команди можуть видаватися на виконання не в тому порядку, в якому вони розташовані в програмі, однак апаратура виявлення і усунення конфліктів між логічно пов'язаними командами забезпечує отримання результатів відповідно до заданої програми.

Відповідна організація комутуючих магістралей, забезпечує засилання результату операції безпосередньо в буфер, який зберігає логічно залежну команду, затриману через

конфлікт, або безпосередньо на вхід функціонального пристрою до того, як цей результат буде записаний в регістровий файл або в пам'ять (short-circuiting, data forwarding, data bypassing - методи, які були розглянуті раніше).

Ще одним апаратним методом мінімізації конфліктів за даними є метод перейменування регістрів (register renaming). Він отримав свою назву від широко вживаного в компіляторах методу перейменування - методу розміщення даних, що сприяє скороченню числа залежностей і тим самим збільшенню продуктивності при відображенні необхідних вихідній програмі об'єктів (наприклад, змінних) на апаратні ресурси (наприклад, комірки пам'яті та регістри).

При апаратній реалізації методу перейменування регістрів виділяються логічні регістри, звернення до яких виконується за допомогою відповідних полів команди, і фізичні регістри, які розміщуються в апаратному регістровому файлі процесора. Номери логічних регістрів динамічно відображаються на номери фізичних регістрів за допомогою таблиць відображення, які оновлюються після декодування кожної команди. Кожний новий результат записується в новий фізичний регістр. Однак попереднє значення кожного логічного регістру зберігається і може бути відновлено у разі, якщо виконання команди має бути перервано через виникнення виняткової ситуації або неправильного передбачення напрямку умовного переходу.

У процесі виконання програми генерується низка тимчасових реєстрових результатів. Ці тимчасові значення записуються в реєстрові файли разом з постійними значеннями. Тимчасове значення стає новим постійним значенням, коли завершується виконання команди (фіксується її результат). У свою чергу, завершення виконання команди відбувається, коли всі попередні команди успішно завершилися в заданому програмою порядку. Програміст має справу тільки з логічними регістрами.

Метод перейменування регістрів спрощує контроль залежностей за даними. У машині, яка може виконувати команди не в порядку їх розташування у програмі, номери логічних регістрів можуть стати двозначними, оскільки один і той же регістр може бути призначений послідовно для зберігання різних значень. Але оскільки номери фізичних регістрів унікально ідентифікують кожен результат, всі неоднозначності усуваються.

5.4. Конфлікти з управління. Скорочення втрат на виконання команд переходу

Конфлікти за управлінням можуть викликати ще більші втрати продуктивності конвеєру ніж конфлікти за даними. Коли виконується команда умовного переходу, вона може або

змінити, або не змінити значення лічильника команд. Якщо команда умовного переходу замінює лічильник команд значенням адреси, обчисленої в команді, то перехід називається виконуваним; в іншому випадку, він називається невиконуваним.

Найпростіший метод роботи з умовними переходами полягає в припиненні конвеєра, як тільки виявлена команда умовного переходу до тих пір, поки вона не досягне ступені конвеєра, яка обчислює нове значення лічильника команд. Такі припинення конвеєра через конфлікти з управління повинні реалізовуватися інакше, ніж припинення через конфлікти за даними, оскільки вибірка команди, наступної за командою умовного переходу, повинна бути виконана як можна швидше, як тільки ми дізнаємося остаточний напрям команди умовного переходу.

Зниження втрат на виконання команд умовного переходу.

Є кілька методів скорочення припинень конвеєра, що виникають через затримки виконання умовних переходів. У даному розділі обговорюються чотири прості схеми, що використовуються під час компіляції. У цих схемах прогнозування напряму переходу виконується статично, тобто прогнозований напрямок переходу фіксується для кожної команди умовного переходу на весь час виконання програми.

1) *Метод вичікування.*

Найпростіша схема обробки команд умовного переходу полягає в заморожуванні операцій в конвеєрі шляхом блокування виконання будь-якої команди наступної за командою умовного переходу, до тих пір, поки не стане відомим напрямок переходу

2) *Метод повернення.*

Більш вдала і не на багато складніша схема полягає в тому, щоб прогнозувати умовний перехід як невиконуваний. При цьому апаратура повинна просто продовжувати виконання програми таким чином, якщо б умовний перехід зовсім не виконувався. У цьому випадку необхідно подбати про те, щоб не змінити стан машини до тих пір, поки напрямок переходу не стане остаточно відомим. У деяких машинах ця схема з невиконуваними за прогнозом умовними переходами реалізована шляхом продовження вибірки команд так, якби умовний перехід був звичайною командою. Поведінка конвеєра виглядає так, як ніби нічого незвичайного не відбувається. Однак, якщо умовний перехід насправді виконується, то необхідно просто очистити конвеєр від команд, обраних слідом за командою умовного переходу і знову повторити вибірку команд з нової адреси.

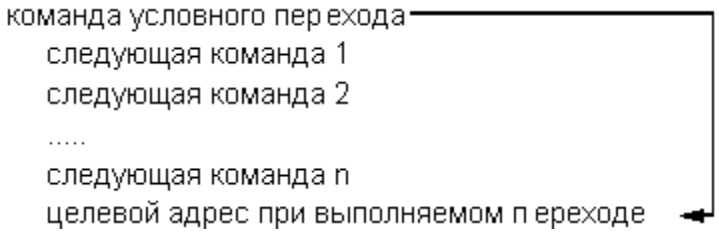
3) Альтернативна схема прогнозує перехід як такий, що виконується. Як тільки команда умовного переходу декодована і обчислено цільову адресу переходу, ми припускаємо, що

перехід виконується, і здійснюємо вибірку команд та їх виконання, починаючи з цільової адреси.

4) *Затримані переходи.*

Четверта схема, яка використовується в деяких машинах, називається «затриманим переходом» та пояснюється наступною схемою. У затриманому переході такт виконання з затримкою переходу завдовжки n є:

Говорять, що команди 1-n перебувають у слотах (тимчасових інтервалах) затриманого



переходу. Завдання програмного забезпечення полягає в тому, щоб зробити команди, які є наступними за командою переходу, дійсно корисними. Апаратура гарантує реальне виконання цих команд перед виконанням безпосередньо переходу. Тут використовуються кілька прийомів оптимізації.

На рис.5.1 показано три випадки, при яких може плануватися затриманий перехід. У

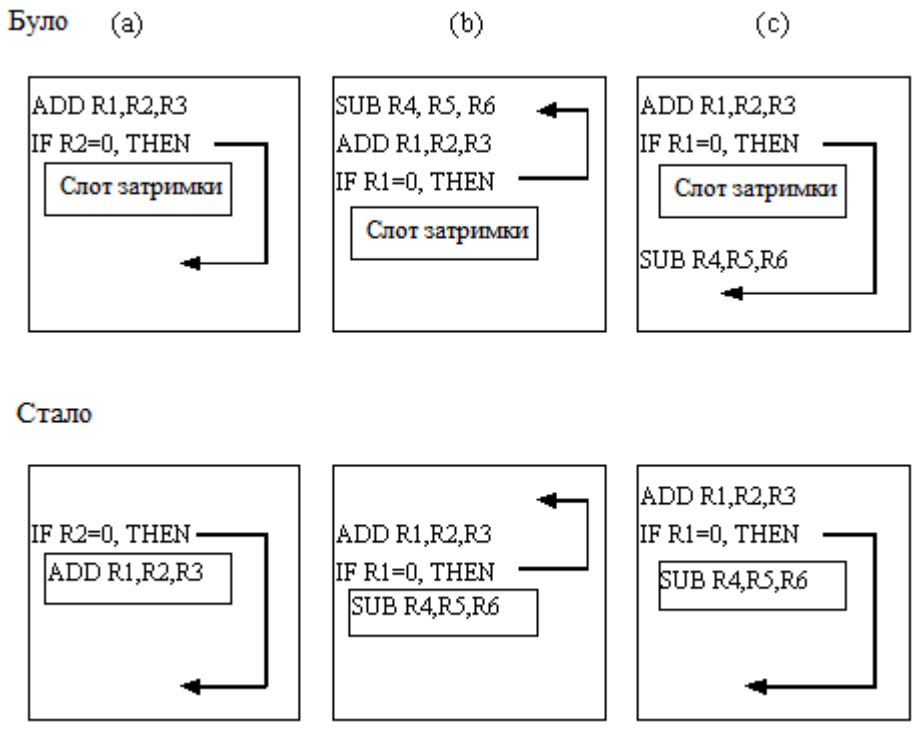


Рис.5.1. Планування затриманих переходів.

верхній частині рисунку для кожного випадку показана вихідна послідовність команд, а в нижній частині - послідовність команд, що отримана в результаті планування. У випадку (a) слот затримки заповнюється незалежною командою, що знаходиться перед командою умовного переходу. Це найкращий вибір. Стратегії (b) і (c) використовуються, якщо застосування стратегії (a) неможливо.

У послідовностях команд для випадків (b) і (c) використання вмісту регістра R1 в якості умови переходу перешкоджає переміщенню команди ADD (яка записує результат в регістр R1) за команду переходу. У випадку (b) слот затримки заповнюється командою, що знаходиться за цільовим адресою команди переходу. Зазвичай таку команду доводиться копіювати, оскільки до неї можливі звернення і з інших частин програми. Стратегії (b) віддається перевага, коли з високою ймовірністю перехід є виконуваним, наприклад, якщо це перехід на початок циклу.

Нарешті, слот затримки може заповнюватися командою, що знаходиться між командою переходу, що не виконується, та командою, що знаходиться за цільовою адресою, як у випадку (c). Щоб подібна оптимізація була законною, необхідно, щоб можна було все-таки виконати команду SUB, якщо перехід піде не за прогнозованим напрямком. При цьому ми припускаємо, що команда SUB виконає непотрібну роботу, але вся програма при цьому буде виконуватися коректно. Це, наприклад, може бути у випадку, якщо регістр R4 використовується тільки для тимчасового зберігання проміжних результатів обчислень, коли перехід виконується не за прогнозованим напрямком.

У табл.5.6 наведено різні обмеження для всіх цих схем планування умовних переходів, а також ситуації, в яких вони дають виграш. Компілятор повинен дотримуватися вимог при підборі підходящої команди для заповнення слоту затримки. Якщо такої команди не знаходиться, слот затримки повинен заповнюватися порожньою операцією.

Статичне прогнозування умовних переходів: використання технології компіляторів.

Є два основні методи, які можна використовувати для статичного прогнозування переходів.

Метод дослідження структури програми. В якості вихідної точки можна припустити, наприклад, що всі переходи, які йдуть назад по програмі, є виконуваними, а ті, що йдуть вперед по програмі,- невиконуваними. Однак ця схема не дуже ефективна для більшості програм. Ґрунтуючись тільки на структурі програми важко зробити кращий прогноз.

Метод використання інформації про профіль виконання програми, який зібраний в результаті попередніх запусків програми. Інформація про профіль програми збирається під час попередніх прогонів. Ключовим моментом, який робить цей підхід заслуговуючим на увагу, є

Табл.5.6.Обмеження при плануванні затриманих переходів.

Випадок	Вимоги	Коли збільшується продуктивність
a)	Команда умовного переходу не повинна залежати від команди, що переставляється.	Завжди.
b)	Виконання команди, що переставляється, має бути коректним, навіть якщо перехід не виконується. Може бути потрібне копіювання команди.	Коли перехід виконується. Може збільшувати розмір програми в разі копіювання команди.
c)	Виконання команди, що переставляється, має бути коректним, навіть якщо перехід виконується.	Коли перехід не виконується.

те, що поведінка переходів при виконанні програми часто повторюється, тобто кожен окремий перехід в програмі часто виявляється зміщеним в одну зі сторін: він або виконується, або не виконується. Проведені багатьма авторами дослідження показують досить успішне прогнозування переходів з використанням цієї стратегії.

Проблеми реалізації точного переривання в конвеєрі.

Обробка переривань у конвеєрній машині виявляється більш складною через те, що поєднане виконання команд ускладнює визначення можливості безпечної зміни стану машини довільною командою. У конвеєрній машині команда виконується по етапах, і її завершення здійснюється через кілька тактів після видачі для виконання. Під час виконання окремих етапів команда може змінити стан машини. Переривання може змусити машину перервати виконання ще не закінчених команд.

Коли відбувається переривання, для коректного збереження стану машини необхідно виконати наступні кроки:

- у послідовність команд, що надходять на обробку в конвеєр, примусово вставити команду переходу на переривання;
- доки виконується команда переходу на переривання, погасити всі вимоги запису, виставлені командою, що викликала переривання, а також всіма наступними за нею в конвеєрі командами. Ці дії дозволяють запобігти змінам стану машини командами, які не завершилися до моменту початку обробки переривання;

- після передачі управління підпрограмі обробки переривань операційної системи, вона негайно повинна зберегти значення адреси команди (PC), що викликала переривання. Це значення буде використовуватися пізніше для організації повернення з переривання.

Якщо використовуються механізми затриманих переходів, стан машини вже неможливо відновити за допомогою одного лічильника команд, оскільки в процесі відновлення команди в конвеєрі можуть виявитися зовсім не послідовними. Зокрема, якщо команда, яка викликала переривання, перебувала у слоті затримки переходу, і перехід був виконаним, то необхідно знову повторити виконання команд з слоту затримки плюс команду, що знаходиться за цільовою адресою команди переходу. Сама команда переходу вже виконалася і її повторення не потрібно. При цьому адреси команд з слоту затримки переходу і цільової адресу команди переходу не є послідовними. Тому необхідно зберігати і відновлювати кілька лічильників команд, число яких на одиницю перевищує довжину слота затримки. Це виконується на третьому кроці обробки переривання.

Після обробки переривання спеціальні команди здійснюють повернення з переривання шляхом перезавантаження лічильників команд і ініціалізації потоку команд. Якщо конвеєр може бути зупинений так, що команди, які безпосередньо передували команді, що викликала переривання, завершуються, а наступні за нею можуть бути заново запущені для виконання, то говорять, що конвеєр забезпечує точне переривання. В ідеалі команда, що викликає переривання, не повинна змінювати стан машини, і для коректної обробки деяких типів переривань потрібно, щоб команда, що викликає переривання, не мала жодних побічних ефектів. Для інших типів переривань, наприклад, для переривань за винятковими ситуаціями з плаваючою точкою, команда що викликає переривання, на деяких машинах записує свої результати ще до того моменту, коли переривання може бути оброблено. У цих випадках апаратура повинна бути готовою для відновлення операндів-джерел, навіть якщо місце

Табл.5.7. Причини переривання на конвеєрі.

Ступінь конвеєра	Причина переривання
IF	Помилка при зверненні до сторінки пам'яті під час вибірки команди; не вирівняне звернення до пам'яті; порушення захисту пам'яті.
ID	Невизначений або заборонений код операції.
EX	Арифметичне переривання.
MEM	Помилка при зверненні до сторінки пам'яті при вибірці даних; не вирівняне звернення до пам'яті; порушення захисту пам'яті.
WB	Відсутній.

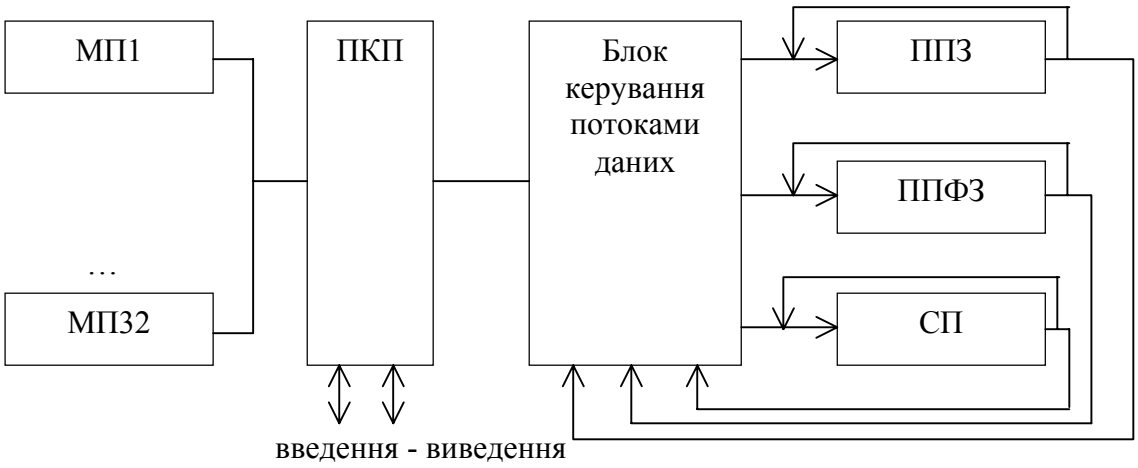


Рис.5.2. Структурна схема системи STAR-100.

розташування результату команди збігається з місцем розташування одного з операндів-джерел.

У табл.5.7 показані ступені розглянутого конвеєра і причини переривань, що можуть виникнути на відповідних етапах при виконанні команд.

Розглянемо приклади систем, що використовують конвеєрну обробку.

У системі STAR-100 (рис.5.2) фірми CDC використовуються як конвеєр команд, так і конвеєр даних. Система містить три конвеєрних процесора:

ППЗ - процесор, що містить конвеєрні пристрої додавання і множення з плаваючою комою;

ППФЗ - процесор, що містить конвеєрний пристрій складання з плаваючою комою, конвеєрний багатоцільовий пристрій, що виконує множення з фіксованою комою, ділення і находження квадратного кореня;

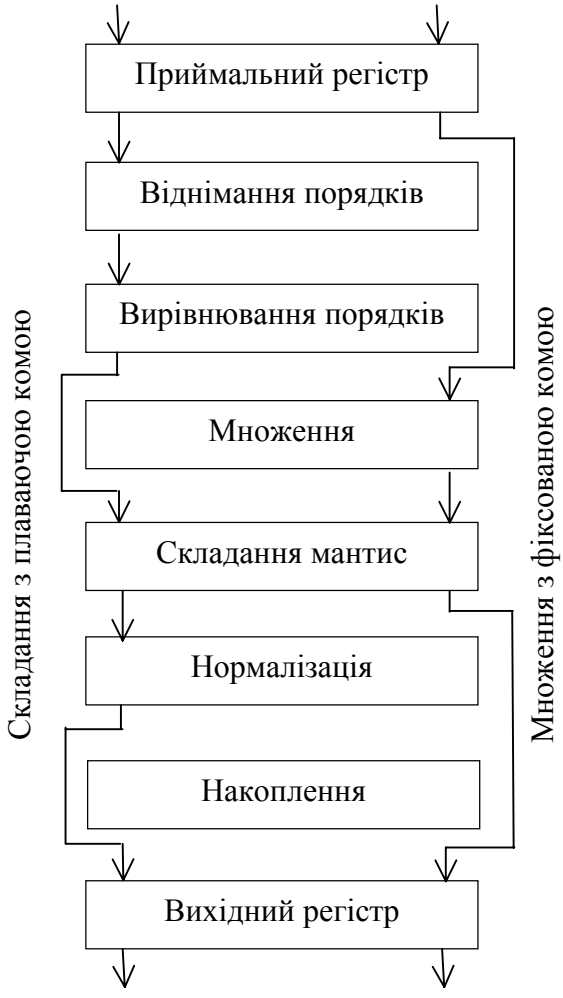


Рис.5.3. Конвейер системы ASC.

СП - спеціальний конвеєрний 16-розрядний процесор, що виконує операції з фіксованою комою і ряд логічних операцій.

Табл.5.8.Порівняння систем паралельної і конвеєрної обробки.

Найменування параметра	Організація	
	Паралельна	Конвеєрна
Базова структура	Незалежне виконання підзадач на окремих блоках апаратури.	Розбиття функцій на N підфункцій.
Продуктивність.	N результатів за T секунд	Один результат за кожні T / N секунд.
Типова архітектура	ОКМД, МКМД.	ОКОД, ОКМД.
Основний період синхронізації	Час для обчислення однієї функції.	Час для одного ступеня конвеєру.
Бажана структура завдань	Матричні задачі з довжинами векторів, які є кратними числу процесорів; процеси, що піддаються розбиттю на незалежні частини.	Переважні одномірні вектори з довільно великою довжиною; забезпечується прискорення виконання традиційного набору команд.
Тимчасова діаграма	Двовимірна (номер процесора і час).	Двовимірна (номер ступені і час).
Типова організація пам'яті	Багаторазово повторені незалежні модулі.	Одна багаторазово розшарована пам'ять.
Типова швидкість доступу до пам'яті	Велика, при коротких періодах часу.	Постійна середня.
Особливості управління	Здійснюється користувачем.	Багато в чому здійснюється апаратурою.
Фактори, що обмежують продуктивність	Вартість, структура завдань.	Елементна база, швидкість доступу до пам'яті.
Забезпечення надійності	Легко досяжна за рахунок «гарячого» резерву.	Обходиться дорого (через немодульну організацію).

Пристрої конвеєрної обробки далеко не завжди виконують з жорстким налаштуванням

на одну певну операцію. Найчастіше їх роблять багатоцільовими, вводячи в конвеєр сегменти, необхідні для реалізації повного набору операцій, а в процесі виконання весь тракт налаштовується відповідним чином. На рис.5.3 наведено структуру системи ASC фірми «Техас Инструментс» і показано, які сегменти універсального конвеєру працюють при різних операціях.

Наприкінці наведемо порівняння конвеєрних систем з системами паралельної обробки даних (табл.5.8).

6. Мультипроцесорні системи. Архітектура, організація обчислювального процесу. Трансп’ютери

6.1. Ступінь зв'язності і основні структури.

Одним із способів класифікації мультипроцесорних систем є ступінь зв'язності її складових частин. На рис.6.1 показано 4 рівня зв'язності, які можуть бути між процесорами. У системах з сильними зв'язками процесори об'єднані за допомогою системної шини, що накладає обмеження на продуктивність системи. Взаємодія через спільну пам'ять є менш сильним видом зв'язності, а для зменшення обмежень, що вносяться загальною шиною, можуть бути застосовані багатопортові запам'ятовуючі пристрої. У разі використання декількох автономних ЕОМ, забезпечених кожна власною операційною системою, об'єднаних у так званий кластер і взаємодіючих за допомогою комунікаційного програмного забезпечення через мережу міжз'єднань, має місце найбільш слабкий рівень зв'язності.

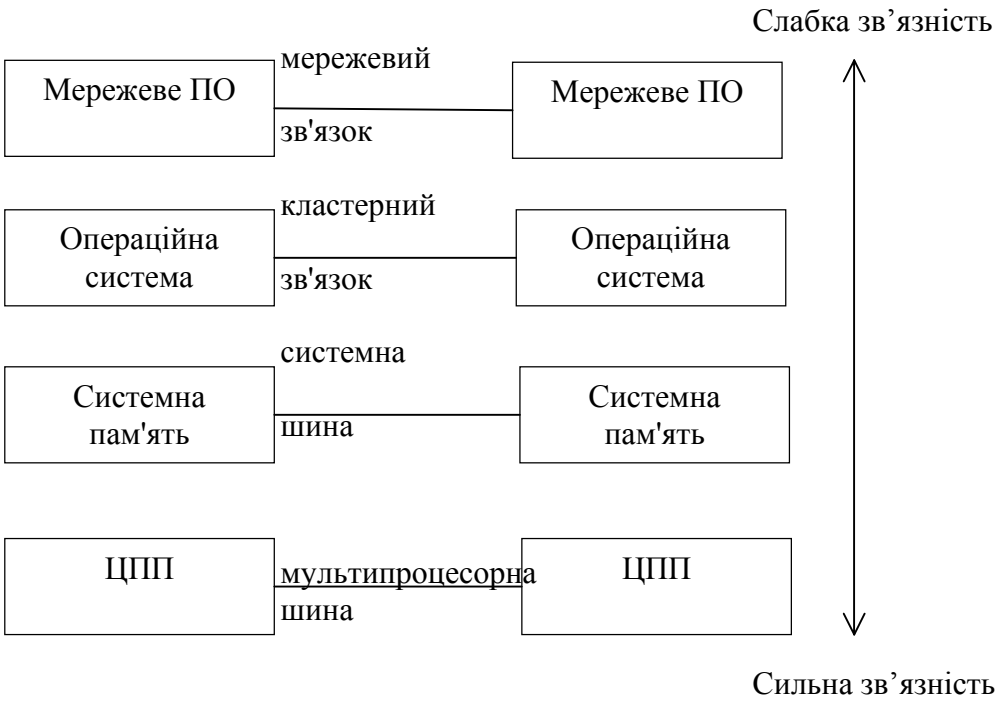


Рис.6.1. Ступені зв'язності мультипроцесорних систем.

За видом взаємовідносин процесорів мультипроцесори можуть бути поділені на системи з автократичним управлінням і системи з рівноправним управлінням. У системах першого типу між процесорами мають місце відносини «хазяїв» та «підлеглих». У системах з

рівноправними процесорами всі вони мають однакові можливості для доступу до загальної шини.

Серед мультипроцесорних систем, що складаються з окремих процесорних та запам'ятовуючих пристроїв, можна виділити системи з конфігурацією типу «танцювальний зал», в яких процесори розміщені в одному ряду, а запам’ятовуючі пристрої звернені у їхній бік - в іншому; з'єднання процесорів з запам'ятовуючими пристроями здійснюється при цьому за допомогою комутаційної мережі (рис.6.2). Іншим крайнім випадком організації мультипроцесорів є конфігурація типу «будуар» (рис.6.3), при якій кожен процесор тісно пов'язаний зі своєю власною пам'яттю, а комутаційна мережа служить для з'єднання між собою пар процесор-пам'ять.

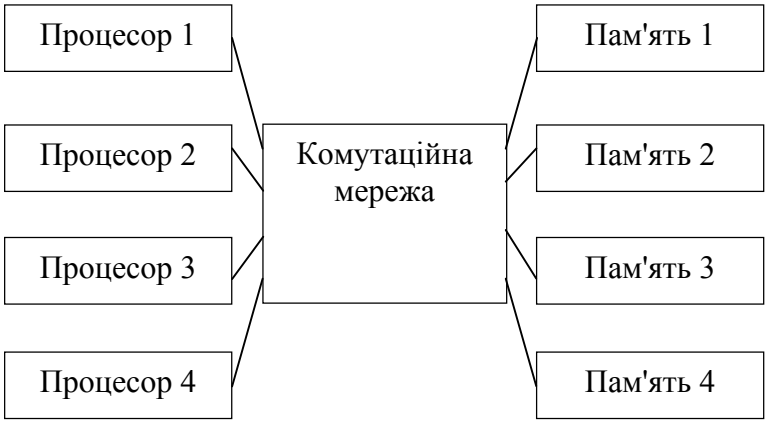


Рис.6.2. Конфігурація «Танцювальний зал».

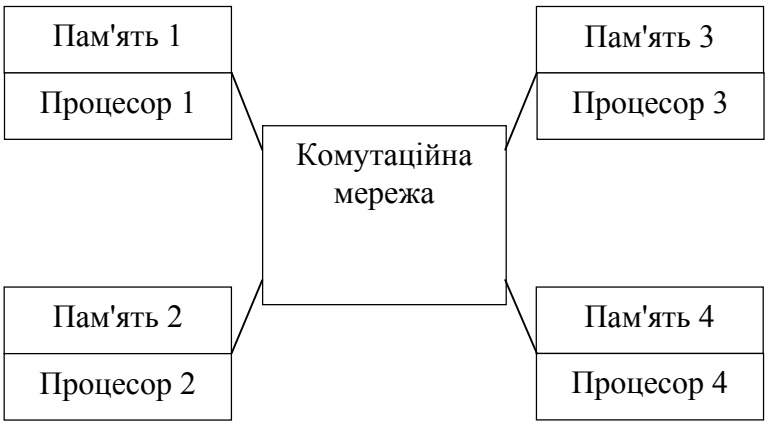


Рис.6.3. Конфігурація «Будуар».

6.2. Моделі зв'язку та архітектури пам'яті.

Однією з відмінних особливостей багатопроцесорної обчислювальної системи є мережа обміну, за допомогою якої процесори з'єднуються один з одним або з пам'яттю. Модель обміну настільки важлива для багатопроцесорної системи, що багато характеристик продуктивності і інші оцінки виражаються відношенням часу обробки до часу обміну, що відповідають вирішуваним задачам. Існують дві основні моделі міжпроцесорного обміну: одна заснована на передачі повідомлень, інша - на використанні загальної пам'яті. У багатопроцесорній системі із загальною пам'яттю один процесор здійснює запис в конкретну комірку, а інший процесор зчитує з цієї комірки пам'яті. Щоб забезпечити узгодженість даних і синхронізацію процесів, обмін часто реалізується за принципом взаємно виключаючого доступу до загальної пам'яті методом «поштової скриньки».

В архітектурах з локальною пам'яттю безпосереднє розділення пам'яті неможливо. Замість цього процесори отримують доступ до спільно використовуваних даних за допомогою передачі повідомлень по мережі обміну. Ефективність схеми комунікацій залежить від протоколів обміну та пропускної здатності пам'яті і каналів обміну.

Модель системи із загальною пам'яттю дуже зручна для програмування і іноді розглядається як високорівневий засіб оцінки впливу обміну на роботу системи, навіть якщо основна система насправді реалізована із застосуванням локальної пам'яті і принципу передачі повідомлень.

У мережах з комутацією каналів і в мережах з комутацією пакетів у міру зростання вимог до обміну слід враховувати можливість перевантаження мережі. Тут міжпроцесорний обмін пов'язує мережеві ресурси: канали, процесори, буфери повідомлень. Обсяг переданої інформації може бути скорочений за рахунок ретельної функціональної декомпозиції задачі і ретельного диспетчирування виконуваних функцій.

Таким чином, існуючі МКМД-машини розпадаються на два основні класи залежно від кількості об'єднуваних процесорів, яка визначає і спосіб організації пам'яті, і методику їх міжз'єднань.

До першої групи відносяться машини із загальною основною пам'яттю, об'єднуючі до декількох десятків (звичайно менше 32) процесорів. Порівняно невелика кількість процесорів в таких машинах дозволяє мати одну централізовану загальну пам'ять і об'єднати процесори і пам'ять за допомогою однієї шини. За наявності у процесорів кеш-пам'яті достатнього обсягу високопродуктивна шина і загальна пам'ять можуть задовольнити звернення до пам'яті, що надходять від декількох процесорів. Оскільки є єдина пам'ять з одним і тим же часом доступу,

ці машини іноді називаються UMA (Uniform Memory Access). Такий спосіб організації з порівняно невеликою пам'яттю, що розділяється в даний час є найбільш популярним. Структура такої системи представлена на рис.6.4.

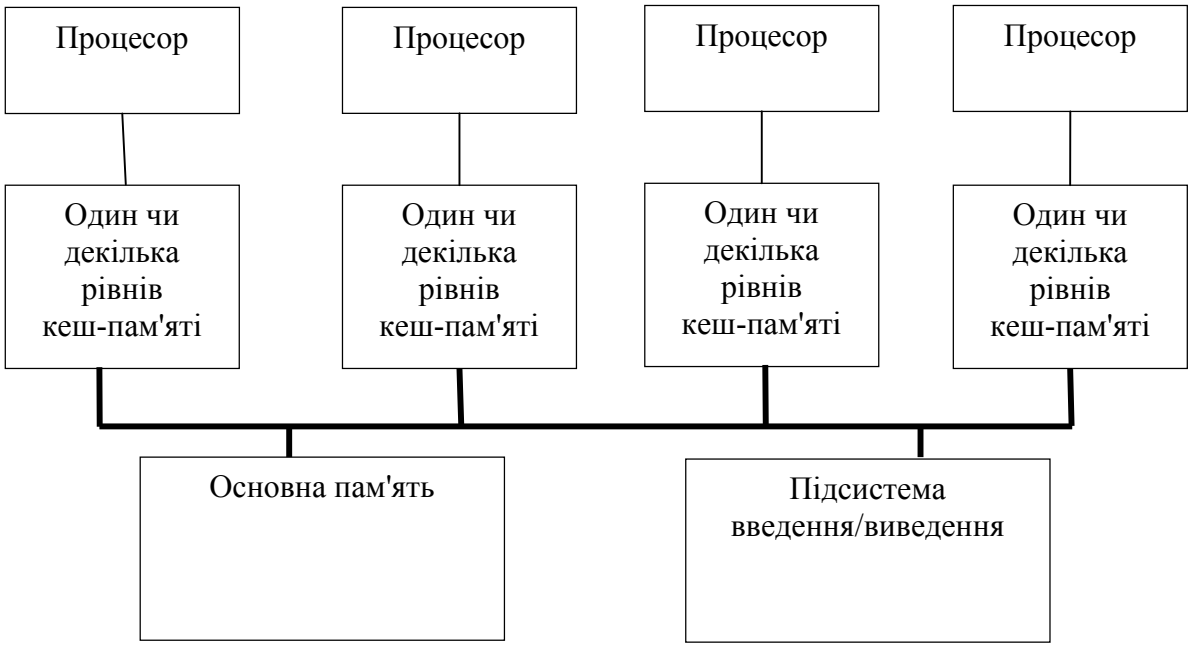


Рис.6.4. Типова архітектура мультипроцесорної системи із загальною пам'яттю.

Другу групу машин складають великомасштабні системи з розподіленою пам'яттю. Для того щоб підтримувати велику кількість процесорів доводиться розподіляти основну пам'ять між ними, в іншому випадку смуги пропускання пам'яті просто може не вистачити для задоволення запитів, що надходять від дуже великої кількості процесорів. Вочевидь, при такому підході також потрібно реалізувати зв'язок процесорів між собою. На рис.6.5 показана структура такої системи.

Зі зростанням числа процесорів просто неможливо обійти необхідність реалізації моделі розподіленої пам'яті з високошвидкісною мережею для зв'язку процесорів. З швидким зростанням продуктивності процесорів і пов'язаним з цим посилюванням вимоги до смуги пропускання пам'яті, масштаб систем (тобто число процесорів в системі), для яких потрібна організація розподіленої пам'яті, зменшується так само, як і зменшується число процесорів, які вдається підтримувати на одній шині та загальній пам'яті.

Розподіл пам'яті між окремими вузлами системи має дві незаперечних переваги. По-перше, це ефективний з точки зору вартості спосіб збільшення смуги пропускання пам'яті, оскільки більшість звернень можуть виконуватися паралельно до локальної пам'яті в кожному

Рис.6.5. Типова архітектура КС з розподіленою пам'яттю.

вузлі. По-друге, це зменшує затримку звернення (час доступу) до локальної пам'яті. Ці дві переваги ще більше скорочують кількість процесорів, для яких архітектура з розподіленою пам'яттю має сенс.

Зазвичай пристрої введення/виведення, як і пам'ять, розподіляються по вузлах та вузли можуть складатися з невеликого числа (2-8) процесорів, з'єднаних між собою іншим способом. Хоча така кластеризація декількох процесорів з пам'яттю і мережевий інтерфейс можуть бути досить корисними з точки зору ефективності у вартісному вираженні, це не дуже суттєво для розуміння того, як така машина працює, тому ми поки зупинимося на системах з одним процесором на вузол. Основна різниця в архітектурі, яку слід виділити в машинах з розподіленою пам'яттю, полягає в тому, як здійснюється зв'язок і яка логічна модель пам'яті використовується.

Як вже було зазначено, будь-яка великомасштабна багатопроцесорна система використовує множину пристроїв пам'яті, які фізично розподіляються разом з процесорами. Є дві альтернативні організації адресації цих пристроїв пам'яті та пов'язані з цим два альтернативні методи для передачі даних між процесорами.

Фізично окремі пристрої пам'яті можуть адресуватися як логічно єдиний адресний простір, що означає, що будь-який процесор може виконувати звернення до будь-яких елементів пам'яті, припускаючи, що він має відповідні права доступу. Такі машини називаються машинами з розподіленою загальною пам'яттю (DSM - distributed shared memory):

масштабовані архітектури з пам'яттю (а іноді NUMA - Non-Uniform Memory Access), оскільки час доступу залежить від розташування комірки в пам'яті.

В альтернативному випадку адресний простір складається з окремих адресних просторів, які логічно не пов'язані, і доступ до яких не може бути здійснений апаратно іншим процесором. У такому прикладі кожен модуль процесор-пам'ять являє собою окремий комп'ютер, тому такі системи називаються багатомашинними (multicomputers).

З кожною з цих організацій адресного простору пов'язаний свій механізм обміну. Для машини з єдиним адресним простором цей адресний простір може бути використаний для обміну даними за допомогою операцій завантаження та запису. Тому ці машини і отримали назву машин з загальною пам'яттю. Для машин з множиною адресних просторів обмін даними повинен використовувати інший механізм: передачу повідомлень між процесорами, тому ці машини часто називають машинами з передачею повідомлень.

Кожен з цих механізмів обміну має свої переваги. Для обміну в загальній пам'яті це включає:

- сумісність з добре зрозумілими механізмами, які використовують для обміну загальну пам'ять;
- простота програмування;
- більш низька затримка обміну і краще використання смуги пропускання при обміні малими порціями даних;
- можливість використання апаратно керованого кешування для зниження частоти віддаленого обміну, що допускає кешування всіх даних (локальних та нелокальних).

Основні переваги обміну за допомогою передачі повідомлень є:

- апаратура може бути більш простою, особливо в порівнянні з моделлю розділеної пам'яті, яка підтримує масштабовану когерентність кеш-пам'яті;
- моделі обміну зрозумілі, примушують програмістів (або компілятори) приділяти увагу обміну, який зазвичай має високу, пов'язану з ним вартість.

Необхідна модель обміну може бути надбудована над апаратною моделлю, яка використовує будь-який з цих механізмів. Підтримка передачі повідомлень над пам'яттю, що розділяється, є набагато простішою, якщо припустити, що машини мають адекватні смуги пропускання. Основні труднощі виникають при роботі з повідомленнями, які можуть бути неправильно вирівняні і повідомленнями довільної довжини в системі пам'яті, яка зазвичай орієнтована на передачу вирівняних блоків даних, організованих як блоки кеш-пам'яті. Ці труднощі можна подолати або з невеликими втратами продуктивності програмним способом, або без втрат при використанні невеликої апаратної підтримки.

Побудова механізмів реалізації колективної пам'яті над механізмом передачі повідомлень набагато складніше. Без передбачуваної підтримки з боку апаратури всі звернення до колективної пам'яті потребують залучення операційної системи як для забезпечення перетворення адрес і захисту пам'яті, так і для перетворення звернень до пам'яті в посилку і прийом повідомлень. Оскільки операції завантаження і запису зазвичай працюють з невеликим об'ємом даних, то великі накладні витрати з підтримання такого обміну роблять неможливою чисто програмну реалізацію.

При оцінюванні будь-якого механізму обміну критичними є три характеристики продуктивності.

Смуга пропускання: в ідеалі смуга пропускання механізму обміну буде обмежена смугами пропускання процесора, пам'яті і системи міжз'єднань, але не аспектами механізму обміну. Накладні витрати, що пов'язані з механізмом обміну (наприклад, довжина міжпроцесорного зв'язку) прямо впливають на смугу пропускання.

Затримка: в ідеалі затримка повинна бути настільки мала, наскільки це можливо. Для її визначення критичними є накладні витрати апаратури і програмного забезпечення, пов'язані з ініціюванням та завершенням обміну.

Маскування затримки: наскільки добре механізм приховує затримку шляхом перекриття обміну з обчисленнями або з іншими обмінами.

Кожен з цих параметрів продуктивності впливає на характеристики обміну. Зокрема, затримка і смуга пропускання можуть змінюватися в залежності від розміру елемента даних. У загальному випадку механізм, який однаково добре працює як з невеликими, так і з великими об'ємами даних буде більш гнучким і ефективним.

Таким чином, відмінності різних машин з розподіленою пам'яттю визначаються моделлю пам'яті і механізмом обміну. Історично машини з розподіленою пам'яттю спочатку були побудовані з використанням механізму передачі повідомлень, оскільки це було очевидно простіше, і багато розробників і дослідники не вірили, що єдиний адресний простір можна побудувати і в машинах з розподіленою пам'яттю. З недавнього часу моделі обміну із загальною пам'яттю дійсно почали підтримуватися практично в кожній розробленої машині (характерним прикладом можуть служити системи із симетричною мультипроцесорною обробкою). Хоча машини з централізованою загальною пам'яттю, побудовані на базі загальної шини, все ще домінують у термінах розміру комп'ютерного ринку, довготривалі технічні тенденції спрямовані на використання переваг розподіленої пам'яті навіть у машинах помірного розміру. Як ми побачимо, можливо, найбільш важливим питанням, яке постає при створенні машин з розподіленою пам'яттю, є питання кешуваня та когерентності кеш-пам'яті.

6.3. Багатопроцесорні системи зі спільною пам'яттю.

Вимоги, що пред'являються сучасними процесорами до смуги пропускання пам'яті, можна істотно скоротити шляхом застосування великих багаторівневих кешів. Тоді, якщо ці вимоги знижуються, то кілька процесорів зможуть розділяти доступ до однієї і тієї ж пам'яті. У такій машині кеші можуть містити як колективні, так і приватні дані. Приватні дані - це дані, які використовуються одним процесором, в той час як колективні дані використовуються багатьма процесорами, по суті забезпечуючи обмін між ними. Коли кешується елемент приватних даних, їх значення переноситься в кеш для скорочення середнього часу доступу, а також необхідної смуги пропускання. Оскільки жоден інший процесор не використовує ці дані, цей процес ідентичний процесу для однопроцесорної машини з кеш-пам'яттю. Якщо кешуються колективні дані, то значення, що буде розділятися, реплікується і може міститися в декількох кешах. Окрім скорочення затримки доступу та необхідної смуги пропускання така реплікація даних сприяє також загальному скороченню кількості обмінів. Однак кешування даних, що розділяються, викликає нову проблему: когерентність кеш-пам'яті.

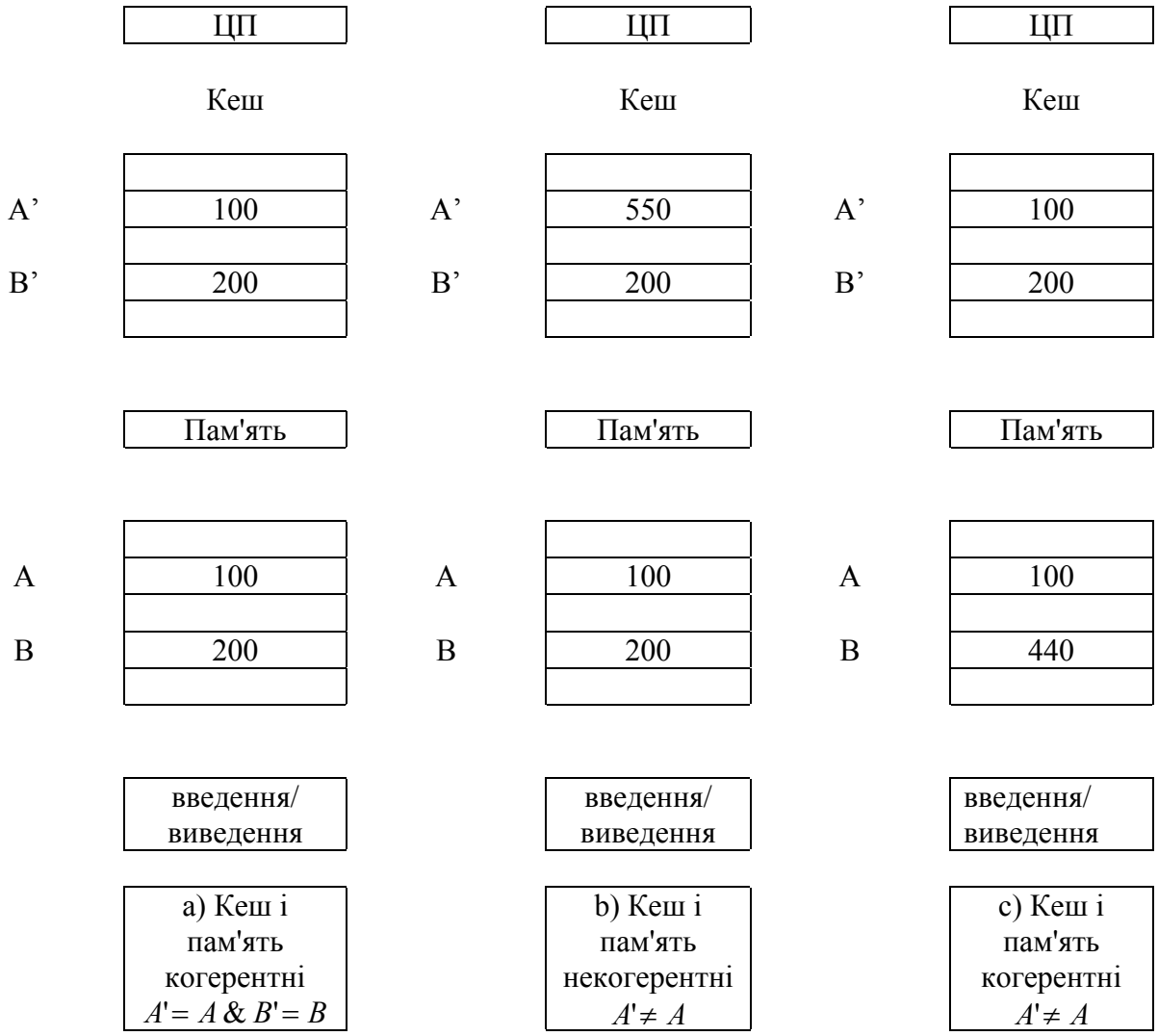
6.4. Мультипроцесорна когерентність кеш-пам'яті.

Проблема, про яку йде мова, виникає через те, що значення елемента даних у пам'яті, яке зберігається у двох різних процесорах, доступно цим процесорам тільки через їх індивідуальні кеші. На рис.6.6 наведено простий приклад, який ілюструє цю проблему.

Проблема когерентності пам'яті для мультипроцесорів та пристроїв введення/виведення має багато аспектів. Зазвичай, в малих мультипроцесорах використовується апаратний механізм, названий протоколом, що дозволяє вирішити цю проблему. Такі протоколи називаються протоколами когерентності кеш-пам'яті. Існують два класи таких протоколів:

Протоколи на основі довідника (directory based). Інформація про стан блоку фізичної пам'яті міститься тільки в одному місці, названому довідником (фізично довідник може бути розподілений по вузлах системи).

Протоколи спостереження (snooping). Кожен кеш, який містить копію даних деякого блоку фізичної пам'яті, має також відповідну копію службової інформації про його стан. Централізована система записів відсутня. Зазвичай, кеші розташовані на загальній (що розділяється) шині і контролери всіх кешей спостерігають за шиною (переглядають її) для визначення того, чи не містять вони копію відповідного блоку.



A’, B’ – кешовані копії елементів A і B в основній пам'яті

a) Когерентний стан кеша та основної пам'яті.

b) Пропонується використання кеш-пам'яті з відкладеним зворотнім копіюванням, коли ЦП записує значення 550 до комірки A. В результаті A’ містить нове значення, а в основній пам'яті залишилося старе значення 100. При спробі виведення A з пам'яті буде отримане старе значення.

c) Підсистема введення/виведення вводить до комірки пам'яті B нове значення 440, а в кеш-пам'яті залишилося старе значення B.

Рис.6.6. Ілюстрація проблеми когерентності кеш-пам'яті.

У мультипроцесорних системах, що використовують мікропроцесори з кеш-пам'яттю, під’єднанні до централізованої загальної пам'яті, протоколи спостереження набули

популярності, оскільки для опитування стану кешей вони можуть використовувати заздалегідь існуюче фізичне з'єднання - шину пам'яті.

Неформально, проблема когерентності пам'яті полягає в необхідності гарантувати, що будь-яке зчитування елемента даних повертає останнє за часом записане в нього значення. Це визначення є не зовсім коректним, оскільки неможливо вимагати, щоб операція зчитування миттєво бачила значення, записане в цей елемент даних деяким іншим процесором. Якщо, наприклад, операція запису на одному процесорі передує операції читання тієї ж комірки на іншому процесорі в межах дуже короткого інтервалу часу, то неможливо гарантувати, що читання поверне записане значення даних, оскільки в цей момент часу записувані дані можуть навіть не залишити процесор. Питання про те, коли точно записуване значення має бути доступне процесору, що виконує читання, визначається обраною моделлю узгодженого (несуперечливого) стану пам'яті і пов'язано з реалізацією синхронізації паралельних обчислень. Тому з метою спрощення припустимо, що ми вимагаємо лише, щоб записане значення було доступно операції читання, що виникла трохи пізніше запису і що операції запису даного процесора завжди видно в порядку їх виконання.

З цим простим визначенням узгодженого стану пам'яті ми можемо гарантувати когерентність шляхом забезпечення двох властивостей:

Операція читання комірки пам'яті одним процесором, що йде за операцією запису в ту ж комірку пам'яті іншим процесором, отримає записане значення, якщо операції читання і запису достатньо відокремлені один від одного за часом.

Операції запису в одну і ту ж комірку пам'яті виконуються строго послідовно (іноді говорять, що вони серіалізовані): це означає, що дві послідовні операції запису в одну і ту ж комірку пам'яті будуть спостерігатися іншими процесорами саме в тому порядку, в якому вони з'являються в програмі процесора, що виконує ці операції запису.

Перша властивість пов'язана з визначенням когерентного (погодженого) стану пам'яті: якби процесор завжди б зчитував лише старе значення даних, ми сказали б, що пам'ять некогерентна.

Необхідність суворо послідовного виконання операцій запису є більш тонкою, але також дуже важливою властивістю. Уявімо собі, що строго послідовне виконання операцій запису не дотримується. Тоді процесор P1 може записати дані в комірку, а потім у цю комірку записує процесор P2. Строго послідовне виконання операцій запису гарантує два важливих наслідки для цієї послідовності операцій запису. По-перше, воно гарантує, що кожен процесор в машині в певний момент часу буде спостерігати запис, виконуваний процесором P2. Якщо послідовність операцій запису не дотримується, то може виникнути ситуація, коли який-

небудь процесор буде спостерігати спочатку операцію запису процесора P2, а потім операцію запису процесора P1, і буде зберігати це записане P1 значення необмежено довго. Більш тонка проблема виникає з підтриманням розумної моделі порядку виконання програм та когерентності пам'яті для користувача: уявіть, що третій процесор постійно читає ту ж саму комірку пам'яті, в яку записують процесори P1 і P2, він повинен спостерігати спочатку значення, записане P1, а потім значення , записане P2. Можливо він ніколи не зможе побачити значення, записаного P1, оскільки запис від P2 виник раніше читання. Якщо він навіть бачить значення, записане P1, він повинен бачити значення, записане P2, при наступному читанні. Подібним чином будь-який інший процесор, який може спостерігати за значеннями, записуваними як P1, так і P2, повинен спостерігати ідентичну поведінку. Найпростіший спосіб добитися таких властивостей полягає в строгому дотриманні порядку операцій запису, щоб всі записи в одну й ту ж саму комірку могли спостерігатися в тому ж самому порядку. Ця властивість називається послідовним виконанням (серіалізацією) операцій запису (write serialization). Питання про те, коли процесор повинен побачити значення, записане іншим процесором досить складне і має помітний вплив на продуктивність, особливо у великих машинах.

6.5. Багатопроцесорні системи з локальною пам'яттю і багатомашинні системи

Існують два різні способи побудови великомасштабних систем з розподіленою пам'яттю. Найпростіший спосіб полягає в тому, щоб виключити апаратні механізми, що забезпечують когерентність кеш-пам'яті, і зосередити увагу на створенні масштабованої системи пам'яті. Кілька компаній розробили такого типу машини. Найбільш відомим прикладом такої системи є комп'ютер T3D компанії Cray Research. У цих машинах пам'ять розподіляється між вузлами (процесорними елементами) і всі вузли з'єднуються між собою за допомогою того чи іншого типу мережі. Доступ до пам'яті може бути локальним або віддаленим. Спеціальні контролери, які розміщуються у вузлах мережі, можуть на основі аналізу адреси звернення прийняти рішення про те, чи знаходяться необхідні дані в локальній пам'яті цього вузла, або розміщуються в пам'яті віддаленого вузла. В останньому випадку контролеру віддаленої пам'яті надсилається повідомлення для звернення до необхідних даних.

Щоб оминути проблеми когерентності, колективні (загальні) дані не кешуються. Звичайно, за допомогою програмного забезпечення можна реалізувати деяку схему кешування даних, що розділяються, шляхом їх копіювання з загального адресного простору в локальну

пам'ять конкретного вузла. У цьому випадку когерентністю пам'яті також управлятиме програмне забезпечення. Перевагою такого підходу є практично мінімальна необхідна підтримка з боку апаратури, хоча наявність, наприклад, таких можливостей як блокове (групове) копіювання даних було б вельми корисним. Недоліком такої організації є те, що механізми програмної підтримки когерентності подібного роду кеш-пам'яті компілятором досить обмежені. Існуюча в даний час методика в основному підходить для програм з добре структурованим паралелізмом на рівні програмного циклу.

Машини з архітектурою, подібною Cray T3D, називають процесорами (машинами) з масовим паралелізмом (MPP Massively Parallel Processor).

Для побудови великомасштабних систем протокол на основі довідника, який відстежує стан кешів. Такий підхід передбачає, що логічно єдиний довідник зберігає стан кожного блоку пам'яті, який може кешуватися. У довіднику зазвичай міститься інформація про те, в яких кешах є копії даного блоку, модифікувався чи даний блок і т.д. В існуючих реалізаціях цього напряму довідник розміщується поряд з пам'яттю. Є також протоколи, в яких частина інформації розміщується у кеш-пам'яті. Позитивною стороною зберігання всієї інформації в єдиному довіднику є простота протоколу, пов'язана з тим, що вся необхідна інформація зосереджена в одному місці. Недоліком такого роду довідників є його розмір, який пропорційний загальному обсягу пам'яті, а не розміру кеш-пам'яті. Це не становить проблеми для машин, що складаються, наприклад, з декількох сотень процесорів, оскільки пов'язані з

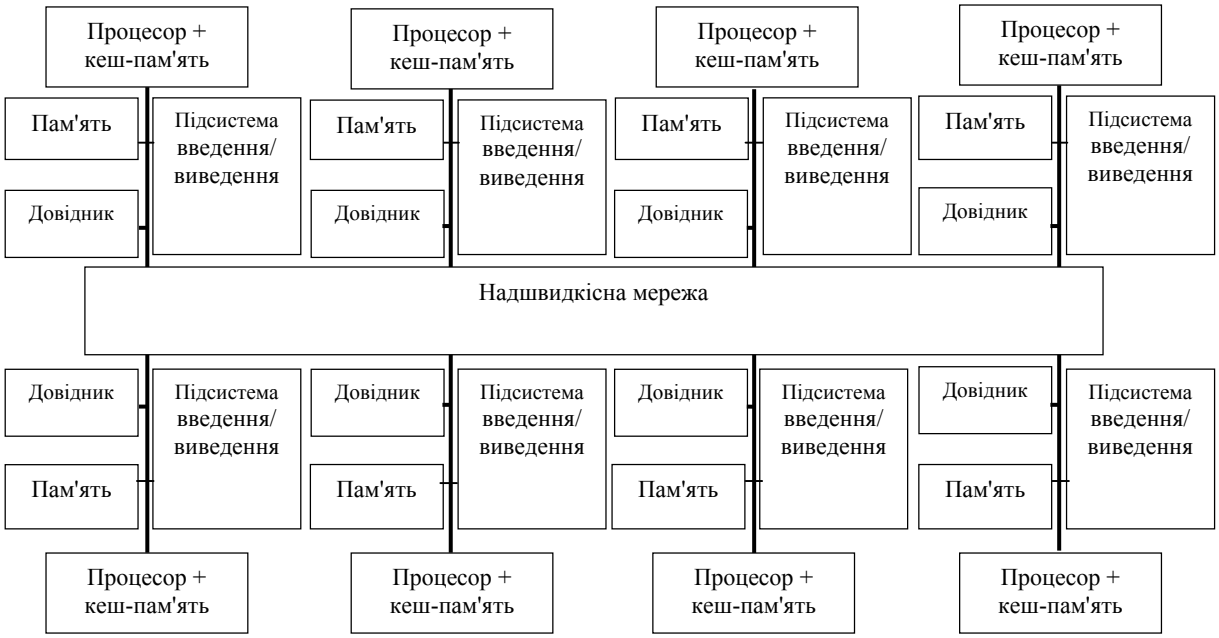


Рис.6.7. Типова архітектура машини з розподіленою пам'яттю.

реалізацією такого довідника накладні витрати можна подолати. Але для машин більшого розміру необхідна методика, що дозволяє ефективно масштабувати структуру довідника.

Зокрема, щоб запобігти появі вузького горла у системі, пов'язаного з єдиним довідником, можна розподілити частини цього довідника разом з пристроями розподіленої локальної пам'яті. Таким чином, можна домогтися того, що звернення до різних довідників (частинам єдиного довідника) можуть виконуватися паралельно, точно так як звернення до локальної пам'яті в розподіленій пам'яті можуть виконуватися паралельно, істотно збільшуючи загальну смугу пропускання пам'яті. У розподіленому довіднику зберігається головна властивість подібних схем, що полягає в тому, що стан будь-якого розділяємого блоку даних завжди знаходиться в цілком певному відомому місці. На рис.6.7 показаний загальний вид подібного роду машини з розподіленою пам'яттю. Питання детальної реалізації протоколів когерентності пам'яті для таких машин виходять за рамки цього огляду.

6.6. Трансп'ютер

Трансп'ютер Inmos T414 призначений для побудови МКМД структур. На рис.6.8 показаний принцип об'єднання трансп'ютерів в тороїдальну матрицю. Для обміну інформацією з іншими процесорами у ньому передбачено чотири швидкодіючих послідовних канали. Є вбудована пам'ять 2Кбайт; локальна пам'ять більшої ємності може бути підключена до інтерфейсу шини пам'яті. Розрядність місцевої пам'яті кожного трансп'ютера нарощує розрядність пам'яті системи; таким чином, повна розрядність пам'яті пропорційна кількості трансп'ютерів в системі. На додаток до паралельної обробки, реалізованої трансп'ютерами, передбачені спеціальні команди для поділу процесорного часу між одночасними процесами.

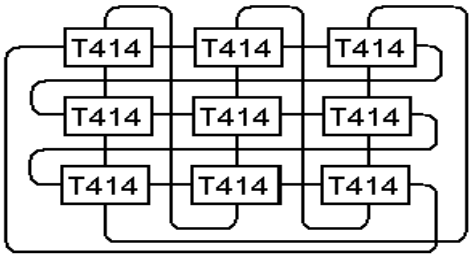


Рис.6.8. Об'єднання трансп'ютерів T414 в тороїдальну матрицю.

Взаємодія кожного трансп'ютера з іншими трансп'ютерами і периферійними пристроями здійснюється за допомогою чотирьох комунікаційних каналів зв'язку, що є в складі БІС (рис.6.9).

Для передачі повідомлень з внутрішньої і внекристальної пам'яті по послідовних каналах застосовується механізм блокових ПДП-пересилань. Інтерфейси зв'язку і процесор працюють одночасно, що дозволяє істотно скоротити втрату продуктивності процесору.

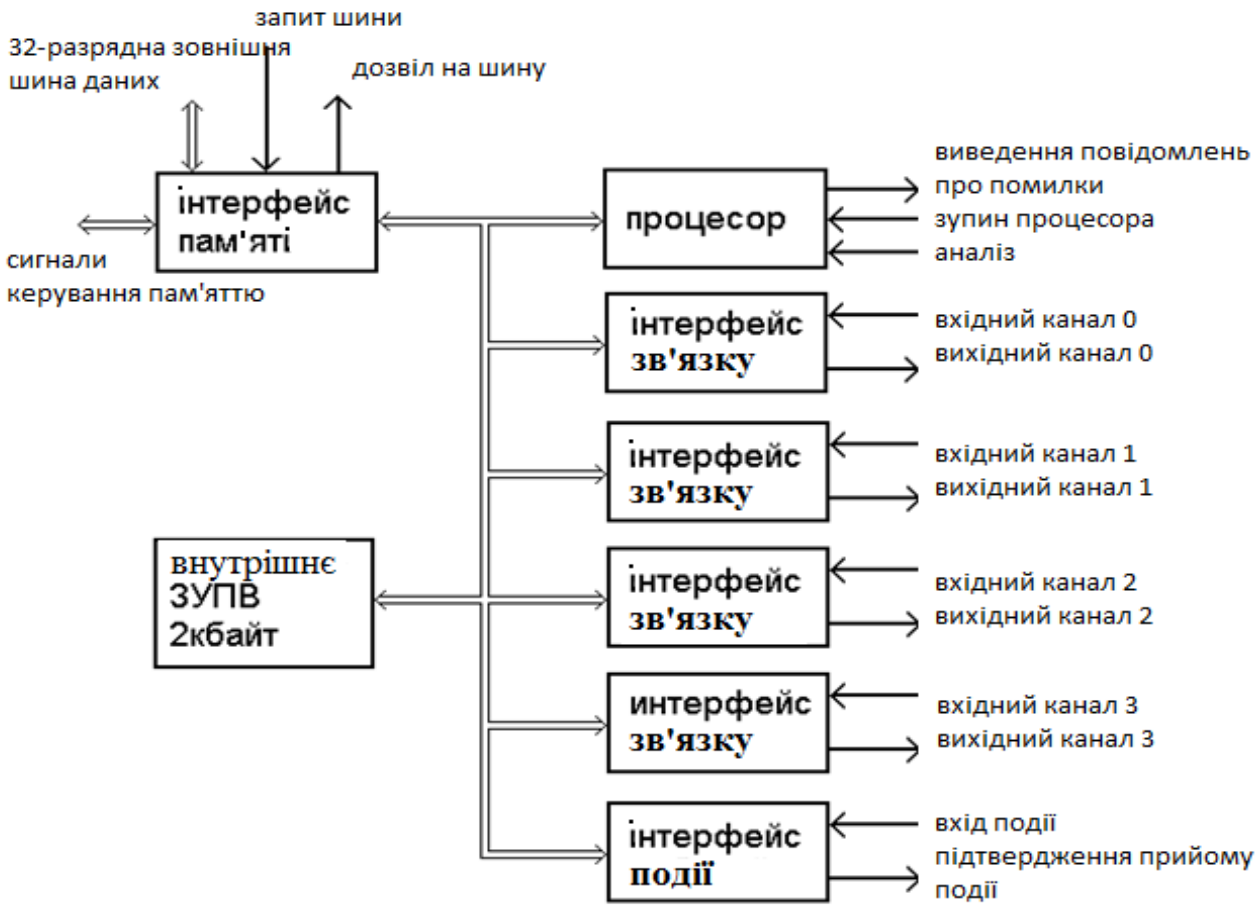


Рис.6.9. Структурна схема трансп’ютера Т414.

Використання прямих послідовних комунікаційних каналів робить непотрібним арбітраж пріоритетів, необхідний для надання шини в мультипроцесорних системах, виключає проблеми, пов'язані з пропускною здатністю шин і їх перевантаженням при введенні в систему нових процесорів.

Кожний послідовний канал дозволяє реалізувати два Оккам-канали, по одному для кожного з двох напрямків передачі. Поки ЦПУ зайнятий, обмін може здійснюватися за всіма чотирма послідовними каналами.

Повідомлення складаються з послідовності байтів. Кожен байт представляється двома одиничними бітами і супроводжується нульовим бітом. У передавальний трансп’ютер подається сигнал підтвердження прийому, що складається з одного одиничного і одного нульового бітів і вказує на те, що приймаючий трансп’ютер готовий до отримання чергового байта. По обом складовим каналу передаються і дані і підтвердження. Щоб пересилання не переривалось, приймаючий трансп’ютер повинен посилати сигнал підтвердження прийому одночасно з початком введення в нього переданого байта.

Трансп’ютер може бути використаний в якості окремого самостійного пристрою, що забезпечує продуктивність 10 млн.операцій/сек. Для повної реалізації можливості об'єднання трансп’ютерів в мережі або матриці при побудові високопродуктивних систем застосовується мова програмування Оккам. При цьому програма, виконувана трансп’ютером, являє собою еквівалент процесу на мові Оккам. Таким чином, трансп’ютерна система безпосередньо описується як програма на мові Оккам. У такій програмі завдання розбивається на підзадачі, що реалізуються одночасно, після чого кожна з цих паралельних підзадач записується у вигляді процесу мови Оккам із зазначенням точок взаємодії процесів. У мультипроцесорних системах засоби мови служать для розподілу навантаження між наявними ресурсами. Одна і та ж програма може виконуватися або на єдиному процесорі, або на декількох за рахунок зміни невеликого числа операторів у заголовку програми.

Трансп’ютер виконує Оккам-програму до тих пір, поки у нього не виникне необхідність отримати додаткову інформацію від інших процесів або в ньому не сформується інформація, яка повинна бути використана іншим процесом. У цих ситуаціях трансп’ютер зупиняє свій процес, запам'ятовує покажчик процесу і переводить процес в режим очікування. Після цього процесор продовжує роботу з іншими процесами, поки не надійде інформація для першого процесу.

Якщо процес реалізується на декількох трансп’ютерах, кожен з них продовжує роботу до тих пір, поки не виявиться готовим до передачі інформації, а потім перебуває в стані очікування до моменту, коли відповідний приймаючий трансп’ютер буде готовий до отримання цієї інформації. Після цього здійснюється пересилка даних і продовжується виконання програми. Будь-які процеси можуть обмінюватися повідомленнями і продовжувати виконуватися як частини повного процесу незалежно від процесів, які перебувають у стані очікування. Описаний механізм зумовлює велику продуктивність трансп’ютерної мережі і високу пропускну здатність її комунікаційних каналів.

Планувальник дозволяє одночасно виконувати будь-яку кількість паралельних процесів, між якими розподіляється процесорний час. Час перемикання процесів складає менше 1мкс, а обмін інформацією між процесами здійснюється з допомогою блокових передач введення-виведення пам'яті. Активні процеси, які очікують виконання, містяться в зв'язній черзі робочих просторів, яка реалізована за допомогою двох регістрів, один з яких вказує на перший процес у черзі, а інший - на останній процес (рис.6.10). Як тільки виконання процесу стає неможливим, покажчик команди запам'ятовується в його робочому просторі, і з черги береться наступний процес.

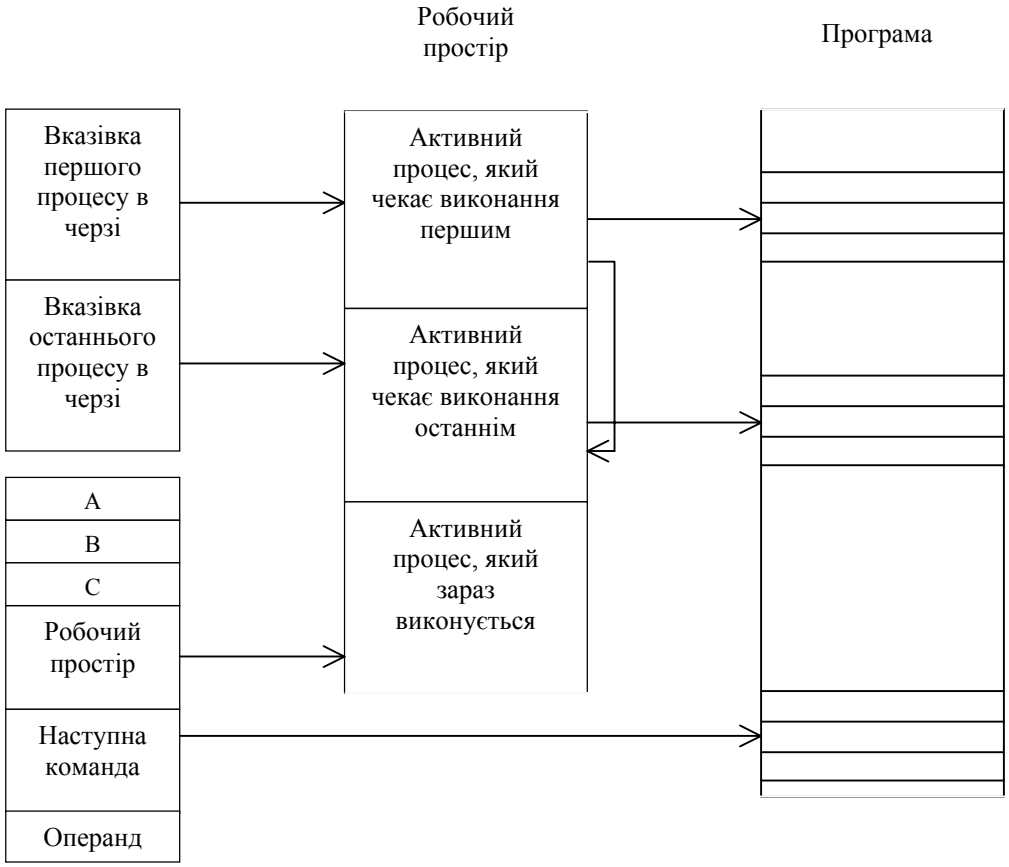


Рис.6.10. Виконання Оккам-програми.

Система переривань в традиційному розумінні відсутня, але мають місце аналогічні засоби, реалізовані у вигляді рівнів пріоритету, які привласнюються процесам, що очікують прийому по послідовних каналах і запитному входу процесу.

Трансп'ютер забезпечує реалізацію дворівневої системи пріоритетів, які позначаються в мові Оккам як PR1 і PAR. Процеси можуть мати високий і низький пріоритет. Процеси низького пріоритету повинні очікувати виконання до тих пір, поки в активному стані не залишиться жодного процесу високого пріоритету. Виконання процесів низького рівня розділяється з певною періодичністю на інтервали часу, щоб процесорний час рівномірно розподілялося між завданнями, обробка яких вимагає великих тимчасових витрат.

Підготовка процесу до введення або виведення полягає у завантаженні в обчислювальний стек:

- ідентифікатора каналу;
- кількості байтів, що підлягають пересиланню;
- покажчика буфера.

6.7. Засоби комплектування ЕОМ. Багатомашинні обчислювальні комплекси

6.7.1. Системи високої готовності

Способи та засоби комплектування ЕОМ розглянемо на прикладі систем високої готовності.

Однією з основних проблем побудови обчислювальних систем у всі часи залишається завдання забезпечення їх тривалого функціонування. Це завдання має три складових: надійність, готовність і зручність обслуговування. Всі ці три складові припускають, в першу чергу, боротьбу з несправностями системи, породжуваними відмовами і збоями у її роботі. Ця боротьба ведеться за всіма трьома напрямками, які взаємопов'язані і застосовуються спільно.

Підвищення надійності ґрунтується на принципі запобігання несправностей шляхом зниження інтенсивності відмов і збоїв за рахунок застосування електронних схем і компонентів із високим і надвисоким ступенем інтеграції, зниження рівня перешкод, полегшених режимів роботи схем, забезпечення теплових режимів їх роботи, а також за рахунок удосконалення методів складання апаратури. Одиницею виміру надійності є середній час напрацювання на відмову (MTBF - Mean Time Between Failure).

Підвищення готовності припускає пригнічення в певних межах впливу відмов і збоїв на роботу системи за допомогою засобів контролю та корекції помилок, а також засобів автоматичного відновлення обчислювального процесу після прояву несправності, включаючи апаратурну і програмну надмірність, на основі якої реалізуються різні варіанти відмовостійких архітектур. Підвищення готовності - це спосіб боротьби за зниження часу простою системи. Одиницею виміру тут є коефіцієнт готовності, який визначає ймовірність перебування системи в працездатному стані в будь-який момент часу. Статистично коефіцієнт готовності визначається як MTBF / (MTBF + MTTR), де MTTR (Mean Time To Repair) - середній час відновлення (ремонту), тобто середній час між моментом виявлення несправності і моментом повернення системи до повноцінного функціонування.

Нижче наведені загальноприйняті в даний час визначення, які ми будемо використовувати для різних типів систем, властивістю яких є та чи інша форма зниження планового і непланового часу простою.

Висока Готовність (High Availability). Реальні конструкції з високим коефіцієнтом готовності для мінімізації планового і непланового часу простою використовують звичайну комп'ютерну технологію. При цьому конфігурація системи забезпечує її швидке відновлення

після виявлення несправності, для чого у ряді місць використовуються надлишкові апаратні і програмні засоби. Тривалість затримки, протягом якої програма, окремий компонент або система простоює, може перебувати в діапазоні від кількох секунд до декількох годин, але більш часто в діапазоні від 2 до 20 хвилин. Зазвичай системи високої готовності добре масштабуються, пропонуючи користувачам більшу гнучкість, ніж інші види надмірності.

Еластичність до відмов (Fault Resiliency). Ряд постачальників комп'ютерного устаткування поділяють весь діапазон систем високої готовності на дві частини, при цьому у верхній його частині виявляються системи еластичні до відмов. Ключовим моментом у визначенні еластичності до відмов є коротший час відновлення, який дозволяє системі швидко відкотитися назад після виявлення несправності.

Стійкість до відмов (Fault Tolerance). Відмовостійкі системи мають у своєму складі надмірну апаратуру для всіх функціональних блоків, включаючи процесори, джерела живлення, підсистеми введення/виведення і підсистеми дискової пам'яті. Якщо відповідний функціональний блок неправильно функціонує, завжди є гарячий резерв. У найбільш просунутих відмовостійких системах надлишкові апаратні засоби можна використовувати для розпаралелювання звичайних робіт. Час відновлення після виявлення несправності для перемикання компонентів, що відмовили, на надлишкові для таких систем зазвичай менше однієї секунди.

Безперервна готовність (Continuous Availability). Вершиною лінії відмовостійких систем є системи, що забезпечують безперервну готовність. Продукт з безперервною готовністю, якщо він працює коректно, усуває будь-який час простою як плановий, так і неплановий. Розробка такої системи охоплює як апаратні засоби, так і програмне забезпечення і дозволяє проводити модернізацію (upgrade) та обслуговування у режимі on-line. Додатковою вимогою до таких систем є відсутність деградації в разі відмови. Час відновлення після відмови не перевищує однієї секунди.

Стійкість до стихійних лих (Disaster Tolerance). Широкий ряд продуктів і послуг пов'язаний із забезпеченням стійкості до стихійних лих. Іноді стійкість до стихійних лих розглядається в контексті систем високої готовності. Сенс цього терміна в дійсності означає можливість рестарту або продовження операцій на іншому майданчику, якщо основне місце розташування системи опиняється в неробочому стані через повені, пожежі або землетрусу. У найпростішому випадку продукти стійкі до стихійних лих можуть просто представляти собою резервні комп'ютери, розташовані поза межею основного розташування системи та сконфігуровані за специфікаціями користувача і доступні для використання в разі стихійного лиха на основному майданчику. У більш складних випадках стійкість до стихійних лих може

означати повне (дзеркальне) дублювання системи поза межею основного розташування, що дозволяє прийняти на себе роботу негайно після відмови системи на основному майданчику.

Всі згадані типи систем високої готовності мають загальну мету - мінімізацію часу простою. Є два типи часу простою комп'ютера: плановий і неплановий. Мінімізація кожного з них вимагає різної стратегії і технології:

а) плановий час простою зазвичай включає час, прийнятий керівництвом, для проведення робіт з модернізації системи і для її обслуговування;

б) неплановий час простою є результатом відмови системи або компонента.

Хоча системи високої готовності можливо більше асоціюються з мінімізацією непланових простоїв, вони виявляються також корисними для зменшення планового часу простою.

Причини планового простою:

- резервне копіювання даних. Деякі конфігурації дискових підсистем високої готовності, особливо системи з дзеркальними дисками, дозволяють проводити резервне копіювання даних в режимі on-line;
- організація робіт з оновлення (модернізації) програмного забезпечення. Сьогодні деякі відмовостійкі системи і всі системи з безперервною готовністю дозволяють проводити модернізацію програмного забезпечення в режимі on-line. Деякі постачальники систем високої готовності також обіцяють такі ж можливості протягом найближчих декількох років.

У загальному випадку, неплановий час простою, перш за все, знижується за рахунок використання надійних частин, резервних магістралей або надмірного устаткування. Однак навіть у цьому випадку система може вимагати достатньо великого планового часу простою.

Спеціальне програмне забезпечення є суттєвою частиною систем високої готовності. При виявленні несправності системи воно забезпечує управління конфігурацією апаратних засобів і програмного забезпечення, а також у разі необхідності процедурами початкової установки, і перебудовує де треба структури даних.

Висока готовність не дається безкоштовно. Загальна вартість подібних систем складається з трьох складових: початкової вартості системи, витрат планування та реалізації, а також системних накладних витрат.

Для реалізації системи високої готовності користувачі повинні на початку закупити власне систему (або системи), що включає один або кілька процесорів в залежності від необхідної обчислювальної потужності і передбачуваної конфігурації; додаткове програмне забезпечення і додатковий дисковий простір.

Щоб реалізувати конфігурацію системи високої готовності найбільш ефективним способом, особливо при використанні кластерних схем, потрібно досить велике попереднє планування. Наприклад, щоб мати можливість перекидання критичного застосування в разі відмови одного процесора на інший, користувачі повинні визначити, які програми є найбільш критичними, проаналізувати всі можливі відмови і скласти докладні плани відновлення на всі випадки відмов.

Накладні витрати систем високої готовності пов'язані з необхідністю підтримки досить складних програмних продуктів, що забезпечують високу готовність. Для забезпечення дублювання записів на дзеркальні диски у разі відсутності спеціальних призначених для цих цілей процесорів, потрібна підтримка додаткової зовнішньої пам'яті.

Вартість системи високої готовності в значній мірі залежить від обраної конфігурації і її можливостей. Нижче наведена деяка інформація, що дозволяє грубо оцінити різні типи надмірності.

Висока готовність. Додаткова вартість систем високої готовності змінюється в межах від 10 до 100 відсотків, зазвичай наближуючись до середини цього діапазону. Додаткова вартість системи високої готовності залежить від того рівня, з яким користувач здатний використовувати резервну систему для обробки своїх додатків. Вартість системи високої готовності може реально перевищити 100 відсотків за рахунок програмного забезпечення та необхідної початкової установки в разі застосування резервної системи, яка не використовується ні для чого іншого. Проте зазвичай резервна система може бути використана для вирішення некритичних завдань, значно знижуючи вартість.

Висока еластичність. Додаткова вартість систем високої еластичності до відмов, що належать до верхнього рівня діапазону систем високої готовності, лежить в межах від 20 до 100 відсотків, знову зазвичай наближуючись до середини цього діапазону. Схеми високої еластичності більш складні і передбачають більш високу вартість планування і великі накладні витрати, ніж системи, що належать нижньому рівню діапазону систем високої готовності. У деяких випадках, однак, користувач може більшою мірою використовувати загальні процесорні ресурси, тим самим, зменшуючи загальну вартість.

Безперервна готовність. Надбавка до вартості для систем з безперервною готовністю знаходиться в діапазоні від 20 до 100 або більше відсотків і зазвичай наближається до верхньої межі цього діапазону. Програмне забезпечення для забезпечення режиму безперервної готовності більш складне, ніж для систем, що забезпечують високу еластичність до відмов. Більшість компонентів системи, такі як процесори, джерела живлення, контролери і кабелі, повинні дублюватися, а іноді і потроюватися. Користувачі систем безперервної готовності, як

і користувачі високо еластичних до відмов систем, мають можливість використати весь набір ресурсів системи велику частину часу, в порівнянні з користувачами більш простих систем, що належать нижньому рівню діапазону систем високої готовності.

Стійкість до стихійних лих. Надбавка до вартості для систем, стійких до стихійних лих, може сильно варіюватися. З одного боку, вона може становити, наприклад, лише кілька відсотків вартості системи при резервуванні часу запасного комп'ютера, що знаходиться поза межею основного майданчика. З іншого боку, вартість системи може збільшитися в кілька разів, якщо необхідно забезпечити дійсно швидке перемикання на іншу систему, що знаходиться на віддаленому майданчику за допомогою високошвидкісних мережевих засобів. Більшість пропозицій по системах, стійким до стихійних лих, вимагають істотного обсягу планування.

Для того, щоб знизити вартість системи, слід ретельно оцінювати дійсно необхідний рівень готовності (тобто здійснювати вибір між високою готовністю, стійкістю до відмов і / або стійкістю до стихійних лих) і вкладати гроші тільки у забезпечення безпеки найбільш критичних для діяльності компанії додатків і даних.

6.7.2. Підсистеми зовнішньої пам'яті систем високої готовності

Першим кроком на шляху забезпечення високої готовності є захист найбільш важливої частини системи, а саме - даних. Різні типи конфігурацій надлишкової зовнішньої пам'яті забезпечують різну ступінь захисту даних і мають різну вартість.

Є три основних типи підсистем зовнішньої пам'яті з високою готовністю. Для своєї реалізації вони використовують технологію Надмірності Масивів Дешевих Дисків (RAID). Найбільш часто використовуються наступні рішення (більш докладно про рівні RAID див. розд. 3.3.3): RAID рівня 1 або дзеркальні диски, RAID рівня 3 з парністю і RAID рівня 5 з розподіленою парністю. Ці три типи зовнішньої пам'яті в загальному випадку мають практично майже миттєвий час відновлення у випадку відмови. Крім того, подібні пристрої іноді дозволяють користувачам змішувати і підбирати типи RAID у межах одного дискового масиву. У загальному випадку дискові масиви представляються прикладній задачі як один диск.

Діапазон можливих конструкцій сучасних дискових масивів досить широкий. Він простягається від простих підсистем без багатьох додаткових можливостей до вельми витончених дискових підсистем, які дозволяють користувачам змішувати і підбирати рівні

RAID всередині одного пристрою. Найбільш потужні дискові підсистеми можуть також містити в своєму складі процесори, які розвантажують основну систему від виконання рутинних операцій введення/виведення, форматування дисків, захисту від помилок і виконання алгоритмів RAID. Більшість дискових масивів обладнані двома портами, що дозволяє користувачам підключати їх до двох різних систем.

Додаток до вартості дискової підсистеми для організації дзеркальних дисків наближається до 100%, оскільки необхідні диски повинні дублюватися в надмірній конфігурації 1:1. Додаткова вартість дисків для RAID рівнів 3 та 5 складає або 33% при наявності диска парності для кожних двох дисків, або частіше 20% при наявності диска парності для кожних чотирьох накопичувачів. Крім того, слід враховувати вартість спеціального програмного забезпечення а також вартість організації самого масиву. Деякі компанії пропонують програмні засоби для організації дзеркальних дисків при використанні звичайних дискових підсистем. Інші пропонують можливості дзеркальної організації в підсистемах RAID, спеціально виготовлених для цих цілей. Підсистеми, що використовують RAID рівнів 3 і 5, відрізняються за вартістю в залежності від своїх можливостей.

Реалізація зовнішньої пам'яті високої готовності може призводити також до збільшення системних накладних витрат. Наприклад, основний процесор системи змушений обробляти дві операції при кожному запису інформації на дзеркальні диски, якщо ці диски не є частиною дзеркального дискового масиву, який має свої власні засоби обробки. Однак найбільш складні дискові масиви дозволяють знизити накладні витрати за рахунок використання процесорів введення/виведення, що є частиною апаратури дискового масиву.

В даний час для пристроїв зовнішньої пам'яті характерна тенденція зменшення співвідношення вартості на одиницю ємності пам'яті (1, 0.5 і менш доларів за мегабайт). Ця тенденція робить сьогоднішні надлишкові рішення по зовнішній пам'яті навіть менш дорогими, у порівнянні з вартістю підсистем зі звичайними дисками, що випускалися тільки рік тому. Тому використання підсистеми RAID дуже швидко стає одним з базових вимог звичайних систем, а не спеціальною властивістю систем високої готовності.

6.7.3. Конфігурація систем високої готовності. Вимоги, що пред'являються до них

В даний час одною з ключових вимог систем високої готовності є можливість їх нарощування з метою забезпечення більш високого ступеня готовності. Головними характеристиками систем високої готовності в порівнянні зі стандартними системами є знижена частота відмов і більш швидкий перехід до нормального режиму функціонування після виникнення несправності за допомогою швидкого відновлення додатків і мережевих сесій до того стану, в якому вони перебували в момент відмови системи. Слід зазначити, що в багатьох випадках користувачів цілком може влаштувати навіть невеликий час простою в обмін на меншу вартість системи високої готовності в порівнянні із значно вищою вартістю забезпечення режиму безперервної готовності.

Конфігурації систем високої готовності, пропоновані сучасною комп'ютерною промисловістю, простягаються в широкому діапазоні від простих жорстких схем, що забезпечують дублювання основної системи окремим гарячим резервом у співвідношенні 1:1, до дуже вільних кластерних схем, що дозволяють одній системі підхопити роботу будь-якої з декількох систем в кластері в разі їх несправності.

Термін «кластеризація» на сьогодні в комп'ютерній промисловості має багато різних значень. Точне визначення могло б звучати так: «реалізація об'єднання машин, що постає в уяві єдиним цілим для операційної системи, системного програмного забезпечення, прикладних програм і користувачів». Машини, кластерізовані разом таким способом, можуть при відмові одного процесора дуже швидко перерозподілити роботу на інші процесори всередині кластеру. Це, можливо, найбільш важливе завдання багатьох постачальників систем високої готовності. Є декілька постачальників, які називають свої системи високої готовності «кластерами» або «простими кластерами», однак на сьогоднішній день реально доступні тільки декілька кластерів, які підпадають під точне визначення (див. нижче).

Сучасні конструкції систем високої готовності припускають використання гарячого резерву (Fail-Over), включаючи перемикання прикладних програм і користувачів на іншу машину з гарантією відсутності втрат або спотворень даних під час відмови і перемикання. У залежності від властивостей системи, деякі або всі ці процеси можуть бути автоматизовані.

Системи високої готовності пов'язані зі своїми резервними системами за допомогою дуже невеликого програмного демона «серцевий пульс», який дозволяє резервній системі керувати основною системою або системами, які вона резервує. Коли «пульс» пропадає, кластер переходить в режим перемикання на резервну систему.

Таке переключення може виконуватися вручну або автоматично, таі є декілька рівнів автоматизації цього процесу. Наприклад, в деяких випадках користувачі інструктуються про те, що вони повинні вийти і знову ввійти в систему. В інших випадках переключення здійснюється більш прозорим для користувача способом: він тільки повинен почекати протягом короткого періоду часу. Іноді користувач може робити вибір між ручним і автоматичним перемиканням. У деяких системах користувачі можуть продовжити роботу після перемикання саме з тієї точки, де вони знаходилися під час відмови. В інших випадках їх просять повторити останню транзакцію.

Резервна система не обов'язково повинна повністю повторювати систему, яку вона резервує (конфігурації систем можуть відрізнятися). Це дозволяє в ряді випадків заощадити гроші за рахунок резервування великої системи або систем за допомогою системи меншого розміру і передбачає або зниження продуктивності у разі відмови основної системи, або переведення на резервну систему тільки критичних для життєдіяльності організації додатків.

Слід додати, що одні користувачі вважають за краще не виконувати жодних додатків на резервній машині, хоча інші навпаки намагаються трохи навантажити резервний сервер в кластері. Можливість вибору конфігурації системи за допомогою процедур початкової установки дає користувачам велику гнучкість, дозволяючи поступово використовувати весь закладений в системі потенціал.

Вимоги початкової установки системи

Більшість систем високої готовності вимагають включення у свій склад процедур початкової установки (System Setup), що забезпечують конфігурацію кластеру для належного виконання процедур перемикання, необхідних у разі відмови. Користувачі можуть запрограмувати «скрипти» початкової установки самостійно або попросити системного інтегратора або постачальника виконати цю роботу. У залежності від того, наскільки складна початкова установка системи, і в залежності від типу системи, з якою мігрує користувач, написання «скриптів», які управляють діями системи високої готовності в разі відмови, може зайняти від одного - двох днів до декількох тижнів або навіть місяців для досвідчених програмістів.

Час простою при перемиканні системи на резервну для систем високої готовності може змінюватися в діапазоні від кількох секунд до 20-40 і більше хвилин. Процедура перемикання на резерв включає в себе наступні етапи:

а) резервна машина виявляє відмову основної і потім виконує розпорядження скрипта, який найімовірніше включає перезапуск системи,

б) передача адрес користувачів;

в) отримання і запуск необхідних додатків, а також виконання певних кроків із забезпечення коректного стану даних.

Час відновлення залежить головним чином від того, наскільки швидко друга машина зможе отримати і запустити програми, а також від того, наскільки швидко операційна система і додатки, такі як бази даних або монітори транзакцій, зможуть отримати приведені в порядок дані.

У загальному випадку апаратне перемикання на резерв займає по порядку величини одну - дві хвилини, а система перезавантажується за наступні одну - дві хвилини. У більшості випадків від 5 до 20 хвилин потрібно на те, щоб отримати і запустити додаток з повністю відновленими даними. В іншому випадку користувачі інструктуються про необхідність заново ввести останню транзакцію.

Додатково до програмного забезпечення систем високої готовності пред'являються наступні вимоги:

- *Журналізація файлової системи*. Коли система відмовляє, журналізірована файлова система гарантує, що файли збережені в останньому узгодженому стані.

- *Ізоляція несправного процесу*. Для активно використовуваних компонентів програмного забезпечення, таких як файлова система, часто застосовується технологія ізоляції несправних процесів, що гарантує ізоляцію помилок в одній системі і неможливість їх розповсюдження за межі цієї системи.

- *Моніторинг обробки транзакцій*. Іноді для керування перемиканням на резерв використовуються монітори транзакцій, що гарантують відсутність втрат даних. При цьому для незавершених транзакцій може бути проведений відкат, тому і бази даних повертаються до відомого узгодженого стану.

6.8 Приклади деякої архітектури обчислювальних систем

6.8.1 Симетрична мультипроцесорна обробка

Symmetric Multiprocessing (SMP). SMP — архітектура суперкомп'ютера, в якій група процесорів працює із загальною оперативною пам'яттю (рис.6.11).

Пам'ять є способом передачі повідомлень між процесорами, при цьому всі обчислювальні пристрої при зверненні до неї мають рівні права і одну і ту ж адресацію для всіх елементів пам'яті.

Роботою управляє єдина копія операційної системи. Для прискорення обробки кожен процесор може також мати власну кеш-пам'ять. Завдання між процесами розподіляються безпосередньо при виконанні прикладного процесу. Навантаження між процесорами динамічно вирівнюється, а обмін даними між ними відбувається з великою швидкістю. Переваги цього підходу полягають в тому, що кожний процесор бачить всю вирішувану задачу в цілому. Але оскільки для взаємодії використовується лише одна шина, то виникають підвищені вимоги до її пропускної спроможності. З'єднання за допомогою шини використовується при невеликому (4—8) числі процесорів.

Рис.6.11. Симетрична мультипроцесорна архітектура

У подібних системах виникає проблема організації когерентності багаторівневої ієрархічної пам'яті.

6.8.2 Асиметрична мультипроцесорна обробка

Asymmetric Multiprocessing (ASMP) — архітектура суперкомп'ютера, в якій кожен процесор має свою оперативну пам'ять.

У ASMP використовуються три схеми (рис. 6.12). В будь-якому разі процесори взаємодіють між собою, передаючи друг другу повідомлення, тобто як би утворюючи швидкісну локальну мережу. Передача повідомлень може здійснюватися через загальну шину (рис. 6.12, а) або завдяки міжпроцесорним зв'язкам. У останньому випадку процесори зв'язані

безпосередньо (рис. 6.12,б), або один через одного (рис. 6.12, в). Безпосередні зв'язки використовуються при невеликому числі процесорів. Кожен процесор має свою оперативну пам'ять, розташовану поряд з ним. Завдяки цьому та якщо це необхідно, процесори можуть розташовуватися в різних, але поруч встановлених корпусах. Група пристроїв в одному корпусі іменується кластером. Користувач, звертаючись до кластера, може працювати відразу з групою процесорів. Таке об'єднання збільшує швидкість обробки даних і розширює використовувану пам'ять. Різко зростає також відмовостійкість, оскільки кластери здійснюють резервне дублювання даних. Створена таким чином система називається кластерною. У цій системі відповідно до її структури може функціонувати декілька копій операційної системи і декілька копій прикладної програми, які працюють з однією і тією ж базою даних (БД), вирішуючи одночасно різні завдання.

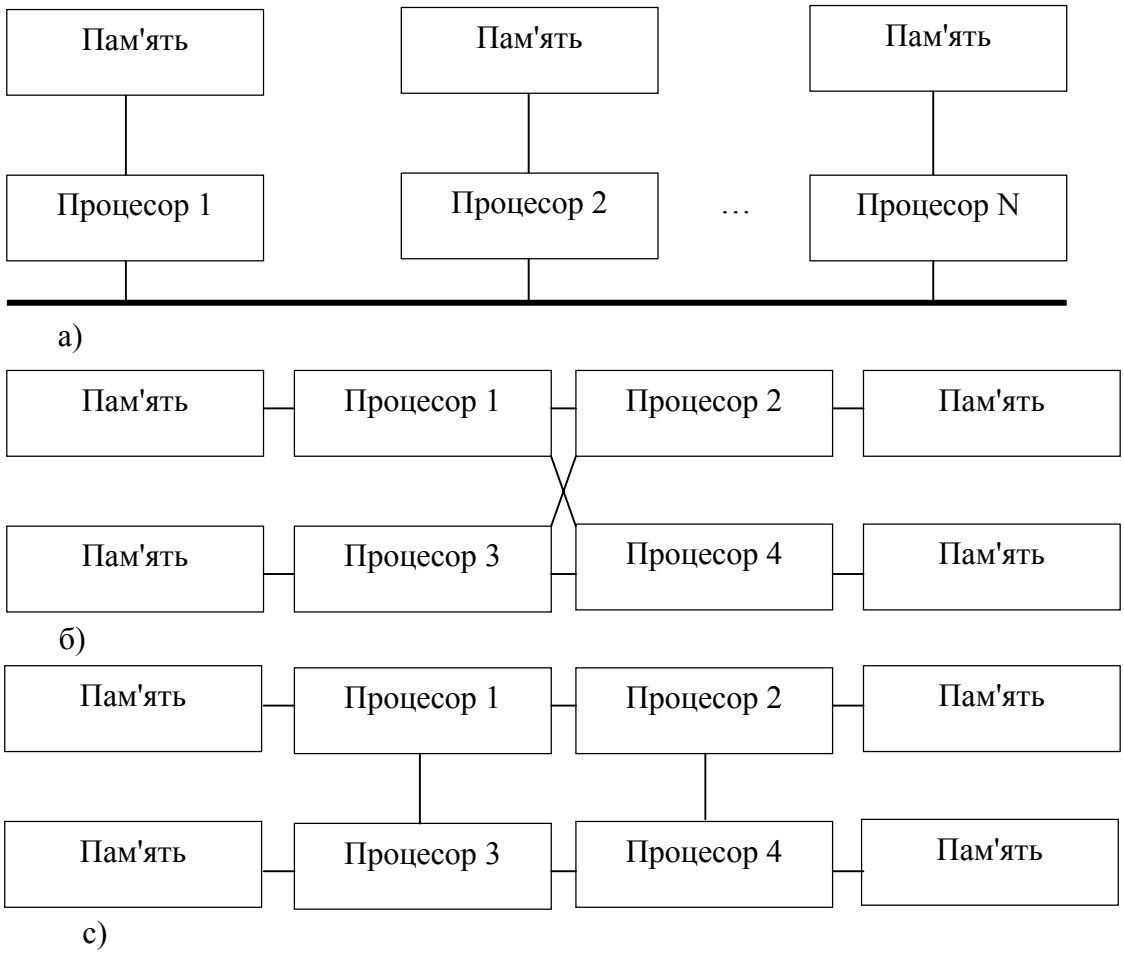


Рис.6.12. Схеми асиметричної багатопроцесорної

6.8.3 Масивна мультипроцесорна архітектура

Massive Parallel Processing — масивно-паралельна архітектура (рис. 6.13, а). В цьому випадку система будується з окремих модулів, кожен з яких містить:

- процесор;

- локальний банк оперативної пам'яті (ОП);

- два комунікаційних процесора (маршрутизатора, роутера — router): один — для передачі команд, інший — для передачі даних (або мережевий адаптер);

- жорсткі диски та інші пристрої введення-виводу.

За своєю суттю, такі модулі є повнофункціональними комп'ютерами. Доступ до банку ОП з даного модуля мають лише процесори з цього ж модуля. Модулі з'єднуються спеціальними комунікаційними каналами. Користувач може визначити логічний номер процесора, до якого він підключений і організувати обмін повідомленнями з іншими процесорами.

6.8.4 Гібридна архітектура

Головна особливість гібридною архітектури NUMA (nonuniform memory access) — неоднорідний доступ до пам'яті.

Гібридна архітектура втілює в собі зручності систем із загальною пам'яттю і відносну дешевизну систем з розподіленою пам'яттю. Суть цієї архітектури — в методі організації пам'яті, а саме: пам'ять є фізично розподіленою по різних частинах системи, але такою, що логічно розділяється, так що користувач бачить єдиний адресний простір. Система складається з однорідних базових модулів (плат), що складаються з невеликого числа процесорів і блоку пам'яті. Модулі об'єднані за допомогою високошвидкісного комутатора. Підтримується єдиний адресний простір, апаратно підтримується доступ до віддаленої пам'яті, тобто до пам'яті інших модулів. При цьому доступ до локальної пам'яті здійснюється у декілька разів швидше, ніж до віддаленої. По суті архітектура NUMA є MPP-архитектурою (масивно-паралельною), де як окремі обчислювальні елементи беруться SMP-вузли.

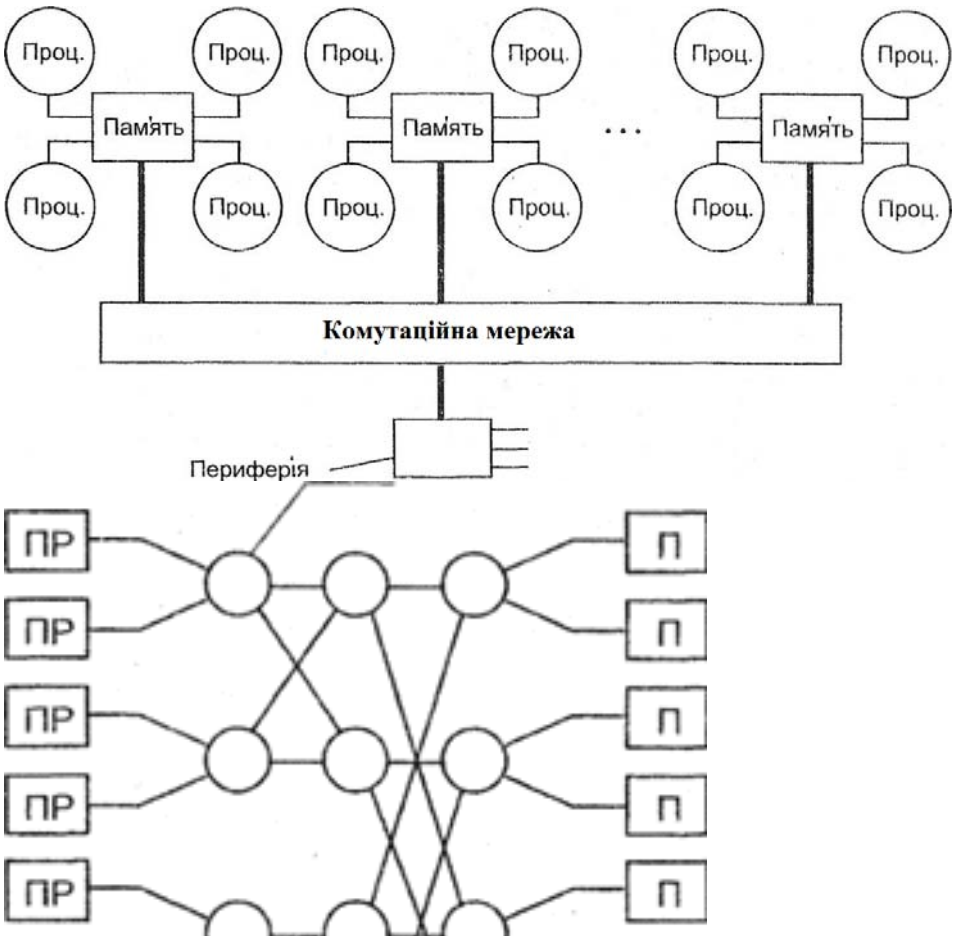


Рис.6.13. Гібридна архітектура

Відомі також гібридні структури з комутатором (рис. 6.13). Тут кожен процесор працює зі своєю пам'яттю, але модулі пристроїв пам'яті зв'язані один з одним за допомогою комутатора (рис. 6.13,а). Комутатори можуть включатися також між групами процесорів (ПР) і модулів пам'яті (П). Тут повідомлення між процесорами і пам'яттю передаються через декілька вузлів (рис. 6.13, б).

6.8.5 Паралельна архітектура з векторними процесорами

Parallel Vector Process — паралельна архітектура з векторними процесорами. Основною ознакою PVP-систем є наявність векторно-конвеєрних процесорів, в яких передбачені команди однотипної обробки векторів незалежних даних, що ефективно виконуються на конвеєрних функціональних пристроях. Як правило, декілька таких процесорів (1—16) працюють одночасно із загальною пам'яттю (аналогічно SMP) в рамках багатопроцесорних

конфігурацій. Декілька таких вузлів можуть бути об'єднані за допомогою комутатора (аналогічно MPP).

6.8.6 Кластерна архітектура

Кластер є двома комп'ютерами або більше (часто званих вузлами), які об'єднуються за допомогою мережевих технологій на базі шинної архітектури. або комутатора і надаються користувачеві як єдиний інформаційно-обчислювальний ресурс. Як вузли кластера можуть виступати сервери, робочі станції або звичайні персональні комп'ютери. Перевага кластеризації для підвищення працездатності стає очевидною в разі збою якого-небудь вузла; при цьому інший вузол кластера може узяти на себе навантаження несправного вузла, і користувачі не помітять переривання в доступі. Можливості масштабованості кластерів дозволяють багато разів збільшувати продуктивність застосувань для більшого числа користувачів.

Типи кластерів:

Тип I. Машина будується цілком із стандартних деталей, які продають багато продавців комп'ютерних компонентів (низькі ціни, просте обслуговування, апаратні компоненти доступні з різних джерел).

Тип II. Система включає ексклюзивні або не широко поширені деталі. Цим можна досягти дуже хорошої продуктивності, проте, при вищій вартості.

6.8.6.1 Зв'язок процесорів в кластерній архітектурі.

Критичним параметром, що впливає на величину продуктивності такої системи, є відстань між процесорами. Так, з'єднавши разом 10 персональних комп'ютерів, можна отримати систему для проведення високопродуктивних обчислень. Проблема, проте, полягатиме в знаходженні найбільш ефективного способу з'єднання стандартних засобів один з одним, оскільки при збільшенні продуктивності кожного процесора в 10 разів продуктивність системи в цілому в 10 разів не збільшиться.

Розглянемо приклад побудови симетричної 16-процесорної системи, в якій всі процесори були б рівноправні. Найбільш природним представляється з'єднання у вигляді плоскої

решітки, де зовнішні кінці можуть використовуватися для під'єднування зовнішніх пристроїв (рис. 6.13).

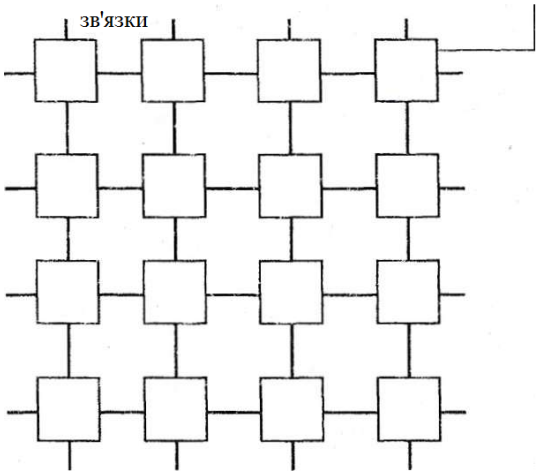


Рис.6.14. . Схема з'єднання процесорів у вигляді грат

При такому типу з'єднання максимальна відстань між процесорами виявиться рівною 6 (кількість зв'язків між процесорами, що відділяють найближчий процесор від найдальшого). Проте, виявляється, що якщо в системі максимальна відстань між, процесорами більше 4, то така система не може працювати ефективно. Для здобуття компактнішої конфігурації необхідно використовувати фігури, що мають максимальний об'єм при мінімальній площі поверхні.

У тривимірному просторі такою властивістю володіє куля. Але оскільки необхідно побудувати вузлову систему, то замість кулі доводиться використовувати куб (якщо число процесорів дорівнює 8, рис.6.15,а) або гіперкуб, якщо число процесорів більше 8. Розмірність гіперкуба визначатиметься залежно від числа процесорів, які необхідно з'єднати. Так, для з'єднання 16 процесорів потрібен чотиривимірний гіперкуб (рис.6.15, б). Для його побудови слід узяти звичайний тривимірний куб, зрушити його в одному напрямі і, з'єднавши вершини, отримати гіперкуб розмірності 4.

Ефективною вважається архітектура з топологією «товстого дерева» (fat-tree). Процесори локалізовані в листі дерева, тоді як внутрішні вузли дерева скомпоновані у внутрішню мережу (рис.6.16). Піддерева можуть спілкуватися між собою, не зачіпаючи вищих рівнів мережі.

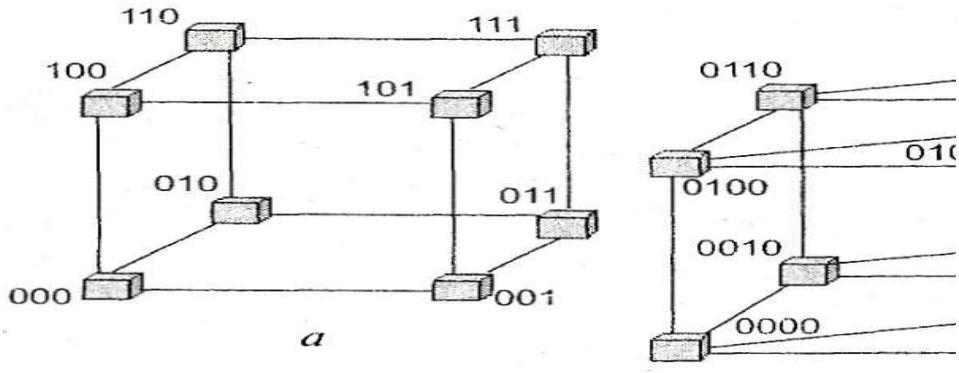


Рис.6.15. Топології зв'язку: а — тривимірний куб; б — чотиривимірний

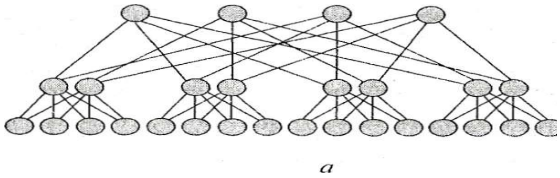


Рис.6.16. Кластерна архітектура «Fat-tree»

Оскільки спосіб з'єднання процесорів один з одним більше впливає на продуктивність кластера, ніж тип використовуваних в ній процесорів, то може виявитися рентабельнішим створити систему з більшого числа дешевих комп'ютерів. У кластерах, як правило, використовуються операційні системи, стандартні для робочих станцій, найчастіше вільно поширювані, — Linux, Free BSD.

Сузір'я

Для певних кластерних конфігурацій останнім часом був запропонований термін сузір'я (constellation).

Розглянемо рис. 6.17. Кожен вузел (node) кластера є незалежним комп'ютером з одним або більше (N) процесорами. Якщо в системі всього M вузлів, причому чисельність вузлів менше цієї кількості (N<M), то має місце кластерна конфігурація, інакше (N>M) маємо справу з сузір'ям.

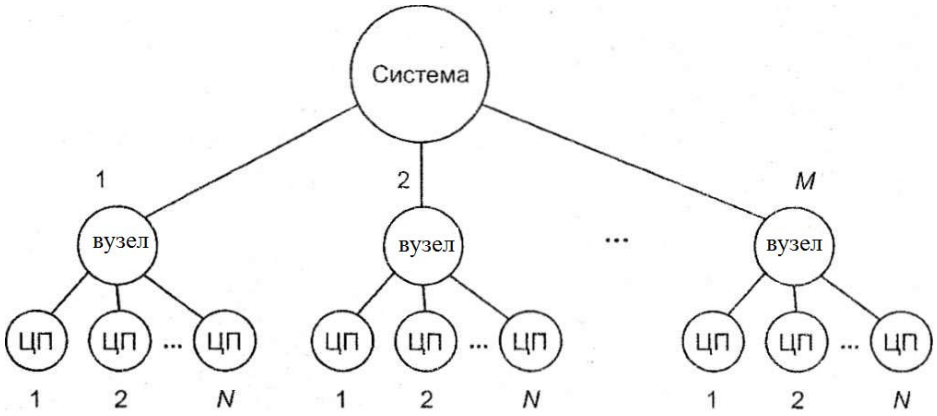


Рис. 6.17. Конфігурації обчислювальних систем: кластери і сузір'я.
У випадку (N > M) маємо справу з сузір'ям.

Система Columbia (кластер з 20 машин SGI Altix, кожна по 512 процесорів) вважається типовим прикладом сузір'я.

7. Основи теорії обчислювальних систем

7.1. Предмет та завдання

Теорія обчислювальних систем - інженерна дисципліна, яка об'єднує методи розв'язання задач проектування та експлуатації ЕОМ, обчислювальних комплексів, систем і мереж.

Предмет теорії. Предметом дослідження в теорії обчислювальних систем є обчислювальні системи в аспектах їх продуктивності, надійності і вартості. У системі виділяють такі складові:

- технічні засоби, які визначаються конфігурацією системи - складом пристроїв і структурою зв'язків між ними;

- режим обробки, що визначає порядок функціонування системи;

- робоче навантаження, що характеризує клас оброблюваних завдань та порядок їх надходження до системи.

Коли ЕОМ, обчислювальний комплекс або система досліджуються в цілому, тобто розглядаються загальносистемні характеристики та властивості, то говорять, що дослідження проводиться на системному рівні.

Завдання аналізу. Аналіз обчислювальних систем - визначення властивостей, притаманних системі або класу систем. Типова задача аналізу - оцінка продуктивності і надійності систем із заданою конфігурацією, режимом функціонування і робочим навантаженням. Інші приклади завдань: визначення (оцінка) ймовірності конфлікту при доступі до загальної шини, розподілу тривалості зайнятості процесора, завантаження каналу вводу-виводу. У загальному випадку задача аналізу формулюється наступним чином. Виходячи з мети дослідження призначається набір характеристик $Y=\{y_1, y_2,\ldots,y_M\}$ досліджуваного об'єкта (обчислювальна система, її елемент, підсистема, певний процес і інш.) і точність $\Delta=\{\delta_1,\delta_2,\ldots,\delta_M\}$, з якими вони повинні бути розподілені. Потрібно знайти спосіб оцінки характеристик Y об'єкта із заданою точністю Δ і на основі цього способу визначити характеристики.

При аналізі системи в процесі експлуатації оцінка характеристик Y виконується, як правило, виміром параметрів функціонування з обробкою вимірювальних даних. У цьому випадку використовується методика, що встановлює склад вимірюваних параметрів, періодичність та тривалість вимірювань, а також вимірювальні засоби і періодичність

вимірювань. З метою скорочення витрат намагаються вимірювати по можливості менше число найбільш просто вимірюваних параметрів $X=\{x_1,x_2,\ldots,x_N\}$, а необхідний набір характеристик визначати непрямим методом - обчисленням за допомогою залежностей $y_m=\varphi_m(X)$, где $m=1,\ldots,M$.

При аналізі проектованих систем для оцінки характеристик Y необхідно володіти моделлю F, яка встановлює залежність Y = F (X) характеристик параметрів системи X, що визначають її конфігурацію, режим функціонування, робоче навантаження. У цьому випадку рішення задачі зводиться до проведення на моделі експериментів, які дозволяють відповісти на питання, що цікавлять. Точність оцінки характеристик проектованої системи залежить від адекватності моделі та похибки вимірювання параметрів X.

Завдання ідентифікації. При експлуатації обчислювальних систем виникає необхідність у підвищенні їх ефективності шляхом підбору конфігурації та режиму функціонування, відповідних класу вирішуваних завдань і вимогам до якості обслуговування користувачів. У зв'язку із зростанням навантаження на систему і переходом на нову технологію обробки даних може знадобитися зміна конфігурації системи, використання більш досконалих операційних систем і реалізованих ними режимів обробки. Побудова моделі системи на основі апріорних відомостей про її організацію і даних вимірювань називається ідентифікацією системи.

Порядок ідентифікації обчислювальної системи показаний на рис.7.1. У відповідності з природою досліджуваних явищ для їх подання пропонується функціональна модель, що

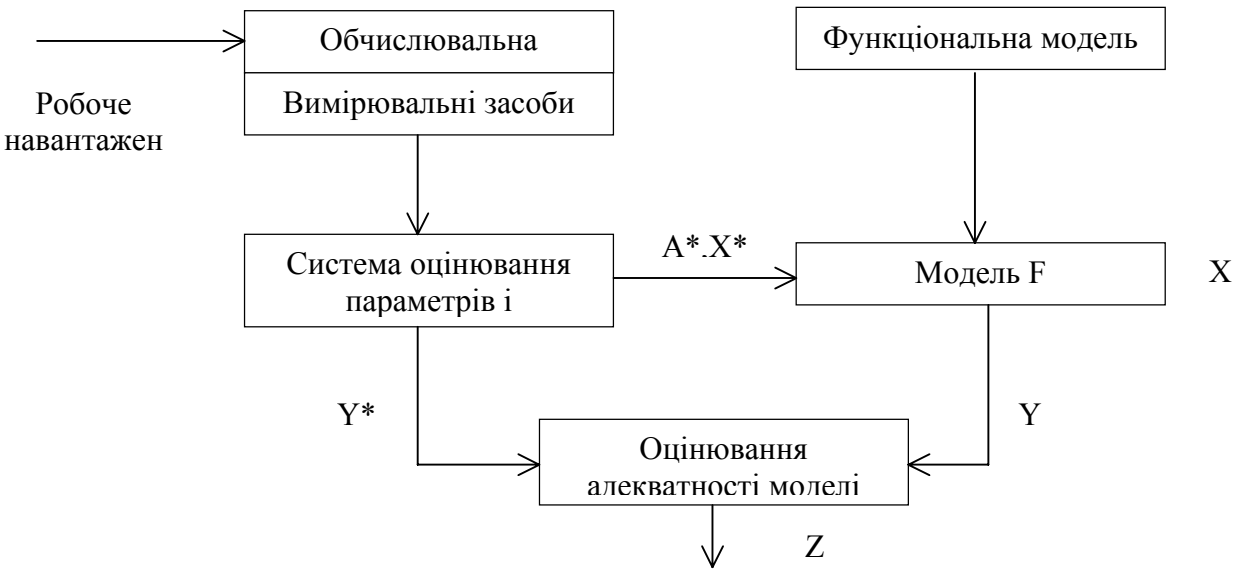


Рис.7.1. Порядок ідентифікації обчислювальної системи.

описує явища з точністю до значень параметрів функцій. Процес створення такої моделі називається функціональною ідентифікацією системи. У ролі функціональних моделей можуть використовуватися різні математичні системи: диференціальні і алгебраїчні рівняння, мережі масового обслуговування та інші, які адекватно представляють досліджувані аспекти.

Завдання синтезу. Синтез - процес створення обчислювальної системи, що найкраще відповідає своєму призначенню. Вихідними в задачі синтезу є такі відомості, що характеризують призначення системи:

1) функція системи (клас розв'язуваних завдань);

2) обмеження на характеристики системи, наприклад на продуктивність, час відповіді, надійність та інші;

3) критерій ефективності, який встановлює спосіб оцінки якості системи в цілому.

Необхідно вибрати конфігурацію системи і режим обробки даних, що задовольняють заданим обмеженням і оптимальні за критерієм ефективності. Типова постановка задачі синтезу: спроектувати систему, що забезпечує рішення заданого класу задач A, з продуктивністю не менше Λ завдань на годину, з середнім напрацюванням на відмову не менш T_0 і мінімальною вартістю.

Математично задача синтезу обчислювальної системи формулюється наступним чином. Нехай $\vartheta=(\varphi_1,\varphi_{2,\ldots},\varphi_Q)$ - вектор параметрів, що характеризують клас задач A, рішення яких є функцією системи; $S = (s_1, \ldots, s_p)$ - вектор параметрів, що характеризує конфігурацію (структуру) системи; $C = (c_1, \ldots, c_r)$ - вектор параметрів режиму обробки; $Y=(y_1,\ldots,y_M)$ - вектор характеристик системи, пов'язаний з параметрами завдань ϑ, конфігурації S і режиму обробки C залежністю $Y=F(\vartheta,S,C)$; $C=\{C_j\}$ - множина можливих режимів обробки. Обмеження на характеристики і параметри системи $z_\alpha,\ldots,z_\omega \in (Y\cup X)$ будемо представляти у вигляді $z_\alpha \in z^*_\alpha,\ldots, z_\omega \in z^*_\omega$ - області допустимих значень відповідних характеристик і параметрів. Критерій ефективності системи представляється заданою функцією $E=\Phi(Y)$, що залежить від характеристик системи, які в свою чергу зумовлюються її параметрами $Y=F(\vartheta,S,C)$. У встановлених позначеннях задача синтезу обчислювальної системи формулюється так: визначити конфігурацію S і режим обробки C, що максимізують ефективність системи:

$$\max \quad E = \max_{S \in S, C \in C} \Phi(Y) \qquad (1)$$

при виконанні обмежень

$$z_\alpha \in z^*_\alpha,\ldots, z_\omega \in z^*_\omega \qquad (2)$$

На відміну від завдання аналізу, спрямованої на визначення характеристик системи Y за заданими параметрами X, завдання синтезу полягає у визначенні параметрів конфігурації S і режиму обробки C, відповідних параметрам робочого навантаження ϑ і характеристикам системи Y, заданих у вигляді (1), (2) . Для задач синтезу характерні два наступних моменти. По-перше, передбачається наявність моделі Y=F(ϑ,S,C), що встановлює залежність характеристик системи від її параметрів. По-друге, завдання синтезу являє собою оптимізаційну задачу і припускає використання методу оптимізації, відповідного виду цільової функції (1) і обмеженням (2). Метод оптимізації повинен гарантувати визначення глобального оптимуму цільової функції E=Φ(Y), визначеної на множині конфігурацій S і режимів обробки C.

7.2. Моделі та методи.

У теорії обчислювальних систем більша увага приділяється моделям продуктивності і надійності і методам забезпечення необхідної продуктивності і надійності при проектуванні та експлуатації систем різного призначення, оскільки вони є найбільш масовими.

Принципи побудови та властивості моделей. Модель - фізична чи абстрактна система, що адекватно представляє об'єкт дослідження. У теорії обчислювальних систем використовуються переважно абстрактні моделі - опису об'єкта дослідження на деякій мові. Абстрактність моделі виявляється в тому, що компонентами моделі є не фізичні елементи, а поняття, у якості яких найбільш широко використовуються математичні. Абстрактна модель, представлена на мові математичних відносин, називається математичною моделлю. Математична модель має форму функціональної залежності Y = F (X), де $Y=\{y_1,y_2,\ldots,y_M\}$ і $X=\{x_1,x_2,\ldots,x_N\}$ - відповідно характеристики і параметри системи і F - функція, відтворювана моделлю. Побудова моделі зводиться до виявлення функції F і поданням її у формі, придатній для обчислення значень Y = F (X). Модель дозволяє оцінювати характеристики Y для заданих параметрів X і вибирати значення параметрів, що забезпечують потрібні характеристики, з використанням процедур оптимізації.

Модель створюється виходячи з мети дослідження, що встановлює:

- склад відтворюваних характеристик $Y=\{y_1,y_2,\ldots,y_M\}$;
- склад параметрів $X=\{x_1,x_2,\ldots,x_N\}$, зміна яких повинна впливати на характеристики Y;
- область зміни параметрів $x_n \in x^*_n$, n=1,…,N, - область визначення моделі;
-точність – гранична припустима похибка оцінки характеристик Y на основі моделі.

Список рекомендованої літератури.

1. Д.Е. Иванов. Вычислительные системи. Методические указания (курс лекций) (для студентов специальности 7.091503 – «Специализированные компьютерные системы»).- Донецк, ДонНТУ, 2002- 98с.
2. Партыка Т.Л., Попов И.И. Электронные вычислительные машины и системы: учебн.пособие.- М.: Форум: ИНФРА-М, 2007.- 368 с.
3. Степанов А.Н. Архитектура вычислительных систем и компьютерных сетей.- СПб.: Питер, 2007.- 509с.
4. Шнитман В. Современные высокопроизводительные компьютеры.- http://citforum.ru/hardware/svk/contents.shtml
5. Богданов А.В., Корхов В.В., Мареев В.В., Станкова Е.Н.Архитектуры и топологии многопроцессорных вычислительных систем.- ИНТУИТ.- 2004.
6. Барановская Т.П., Лойко В.И., Семёнов М.И., Трубилин А.И. Архитектура компьютерных систем и сетей.- Москва: Финансы и статистика, 2003.- 256с.
7. Фельдман Л.П., Дедищев В.А. Математическое обеспечение САПР. Моделирование вычислительных и управляющих систем.- Киев:УМК ВО, 1992.
8. Основы теории вычислительных систем // Под редакцией Салыги В.И.- Харьков: Издательство при Харьковском государственном университете, 1984.- 200с.
9. Каган Б.М. Электронные вычислительные машины и системы.- Москва: Энергоатомиздат, 1991.- 592с.
10. Ларионов А.М., Майоров С.А., Новиков Г.И. Вычислительные комплексы, системы и сети.- Ленинград: Энергоатомиздат, 1987.- 288с.
11. Основы теории вычислительных систем / под. ред. С.А. Майорова.- Москав: Высшая школа.- 1978.- 408с.
12. Фрир Дж. Построение вычислительных систем на базе перспективных микропроцессоров.- Москва: Мир, 1990.- 413с.
13. Коуги П.М. Архитектура конвейерных ЭВМ.- Москва: Радио и связь, 1985.- 360с.
14. Ю.С. Затуливетер, Е.А. Фищенко Компьютер ПС-2000: Многопроцессорная архитектура, опередившая время // Пленарные и избранные доклады 4-й Международной конференции «Параллельные вычисления и задачи управления», Москва, 27-29 октября 2008.