Тема 2. Інформаційні технології організації високопродуктивних обчислень

1. Організація паралельних обчислень з використанням сучасних технологій.
2. Класифікація сучасних паралельних обчислювальних систем.
3. Рівні паралелізму.
4. Комбінування паралельних та розподілених обчислень.
5. Високопродуктивні обчислювальні системи з гібридною архітектурою.
6. Технологія GPGPU.

1 Організація паралельних обчислень з використанням сучасних технологій

Останні десятиліття високопродуктивні обчислювальні системи знаходять своє застосування при вирішенні практично будь-яких завдань науки і техніки в усіх галузях народного господарства. Серед таких завдань – моделювання різних фізичних процесів, задачі обчислювальної хімії та біології, нанотехнології, автоматизація проектування та багато інших. Прогрес у галузі високопродуктивних обчислень багато в чому визначає темп розвитку науки і техніки, і, в остаточному підсумку, рівень технологічного розвитку держави в цілому. Тому, можна з упевненістю стверджувати, що створення і вивчення методів розробки програмно-апаратного забезпечення для високопродуктивних обчислювальних систем є однією із найважливіших задач сучасних інформаційних технологій.

Паралельні обчислення є узагальненим терміном, що застосовується для позначення технологій та методів розробки програмно-апаратного забезпечення для високопродуктивних комп'ютерних систем. Тому, термін "паралельні обчислення" описує достатньо широку галузь, яка пов'язана з організацією розрахунків в обчислювальних системах, що містять декілька процесорних пристроїв. До таких систем відносять багатоядерні процесори, багатопроцесорні системи із загальною пам'яттю, високопродуктивні обчислювальні кластери з розподіленою пам'яттю або гібридною архітектурою, системи, що реалізують загальні обчислення на основі відеоадаптерів (GPGPU), хмарні обчислення (Cloud Computing), тощо.

У наукових публікаціях, дослідженнях і прикладних проектах паралельним обчисленням останнім часом приділяється велика увага. Це пов'язано, переважно, з двома чинниками. Перший фактор обумовлений науково-технічним прогресом, у результаті якого з'явилися нові галузі знань, що потребують застосування високопродуктивних методів математичного моделювання. Відповідно, і моделі також істотно ускладнились. У підсумку, спостерігається тенденція зростання потреби в ресурсоємних розрахунках, які в ряді випадків можна виконати лише на базі високопродуктивної обчислювальної техніки і виключно за допомогою методів паралельних, розподілених або ж гетерогенних обчислень.

Другий істотний фактор, в результаті якого інтерес до паралельних обчислень суттєво зріс, полягає в широкому розповсюдженні паралельних комп'ютерів. Останнім часом багатопроцесорні сервери можна часто зустріти на середніх і великих підприємствах, у банках, дослідних інститутах та обчислювальних центрах.

З появою багатоядерних процесорів та відеоадаптерів багато користувачів стали володарями своєрідних "міні-суперкомп'ютерів" на своїх робочих місцях. Істотний прогрес у галузі мережевих технологій дозволив використовувати для паралельних обчислень локальні мережі підприємств, навчальні класи освітніх установ, уможливив створення відносно недорогих обчислювальних кластерів.

У підсумку, можна стверджувати, що "паралельні інформаційні технології" перетворилися з вузькоспеціальної дисципліни в необхідну складову інформаційної інфраструктури різноманітних установ та організацій – з одного боку, а з другого боку – комплексу знань фахівців з інформаційних технологій та комп’ютерної інженерії, розробників сучасного програмно-апаратного забезпечення.

Зокрема, актуальним є застосування паралельних обчислень в галузях, що пов'язані з проведенням ресурсоємних та складних розрахунків, а саме:

– ***системах підтримки проектування (CAD – Computer Aided Design).*** У таких системах необхідність здійснювати моделювання в реальному часі висуває високі вимоги до продуктивності програмно-апаратного забезпечення. В результаті ж застосування паралельних інформаційних технологій вдається прискорити процес проектування, і тим самим, знизити часові та трудові витрати на розробку нової моделі;

– ***складних інженерних розрахунках та імітаційному моделюванні.*** До цього класу належать різноманітні задачі з галузі моделювання аварійних ситуацій, моделювання робастних систем, міцнісного моделювання і багато інших;

– ***математичному моделюванні фізичних процесів.*** До цього широкого класу задач відносять сфери дослідження динаміки рідини і газу, електромагнітні і ядерні взаємодії, процеси горіння і т.п. Такі процеси, як правило, описуються системами рівнянь в часткових похідних. Застосування для вирішення таких задач різницевих методів найчастіше потребує дуже великих обсягів обчислень і пам'яті. Використання багатопроцесорних систем і методів паралельних та розподілених обчислень дозволяє підвищити показники точності та продуктивності моделювання;

– ***моделюванні глобальних процесів.*** В першу чергу, це задачі прогнозування зміни клімату, природних катаклізмів, тощо. Також, великою обчислювальною складністю характеризуються різноманітні геологічні дослідження, пов'язані з аналізом будови та процесів у надрах планети;

– ***обчислювальної хімії.*** Різноманітні задачі цієї галузі спрямовані на вивчення властивостей речовини в різних станах. Широке застосування методів молекулярної динаміки також часто потребує значних обчислювальних ресурсів, що підтверджує актуальність застосування паралельного програмування. До даної категорії можна також віднести задачі, пов'язані з оптимальною конфігурацією протеїнів, розшифрування ДНК і багато інших проблем, суміжних з хімічною галуззю;

– ***бізнес-додатках.*** До цієї категорії відносяться задачі, пов'язані з аналізом фінансових ринків і прогнозуванням курсів валют. Також поширені оптимізаційні задачі для формування прогнозу та прийняття рішення щодо найкращого варіанта використання фінансових або інших ресурсів, побудови оптимальних транспортних і телекомунікаційних мереж, розміщення підприємств в регіоні тощо.

Даний перелік можна продовжувати й далі, адже він є достатньо великим. Проте, вище наведено лише деякі, найбільш виражені за своєю актуальністю із численних застосувань систем паралельних обчислень, перелік сфер застосування яких неухильно і активно розширюється.

2 Класифікація сучасних паралельних обчислювальних систем

Навіть короткий перелік типів сучасних паралельних обчислювальних систем (ОС) дає зрозуміти, що для орієнтування в цьому різноманітті необхідна чітка система класифікації. Серед усіх розглянутих систем класифікації ОС найбільше визнання отримала класифікація, запропонована в 1966 році М. Флінном. В її основу покладено поняття потоку, під яким розуміється послідовність елементів, команд або даних, яка обробляється процесором. Залежно від кількості потоків команд і потоків даних Флінн виділяє чотири класи архітектур: SISD, MISD, SIMD, MIMD (табл. 2.1, рис. 2.1).

Таблиця 2.1

Класифікація архітектур комп'ютерних систем

Потік команд	**Одиничний потік даних (ОД)**	**Множинний потік даних (МД)**
Одиничний (ОК)	**ОКОД (SISD)** (однопроцесорні комп'ютери)	**ОКМД (SIMD)** (комп'ютери з паралельними або асоціативними процесорами)
Множинний (МК)	**МКОД (MISD)** (конвеєрні магістральні комп'ютери)	**МКМД (MIMD)** (багатопроцесорні або багатомашинні комплекси)

SISD (Single Instruction Stream/Single Data Stream) - одиничний потік команд і одиничний потік даних. Представник цього класу – класичний фон-нейманівський комп'ютер. Команди обробляються послідовно і кожна команда ініціює одну операцію з одним потоком даних. До цього класу комп'ютерів можна віднести також конвеєрні комп'ютери. Деякі спеціалісти вважають, що до SISD-систем можна віднести і векторно-конвеєрні ОС, якщо розглядати вектор як неподільний елемент даних для відповідної команди.

MISD (Multiple Instruction Stream/Single Data Stream) – множинний потік команд і одиничний потік даних. В архітектурі присутня множина процесорів, які обробляють один і той же потік даних. Ряд дослідників відносять до даного класу конвеєрні системи. Прийнято вважати, що доки даний клас незадіяний, однак може бути корисним для розробки принципово нових концепцій побудови обчислювальних систем.

SIMD (Single Instruction Stream/Multiple Data Stream) – одиничний потік команд і множинний потік даних. Ця архітектура дозволяє виконати одну арифметичну операцію відразу над багатьма даними – елементами вектору. Представниками цього класу є системи з матрицею процесорів, де один управляючий пристрій контролює множину процесорних елементів. Усі

процесорні елементи отримують від пристрою управління однакову команду і виконують її над власними локальними даними. В цей клас можуть бути включеними і векторно-конвеєрні ОС, якщо кожний елемент вектора розглядати як окремий елемент даних.

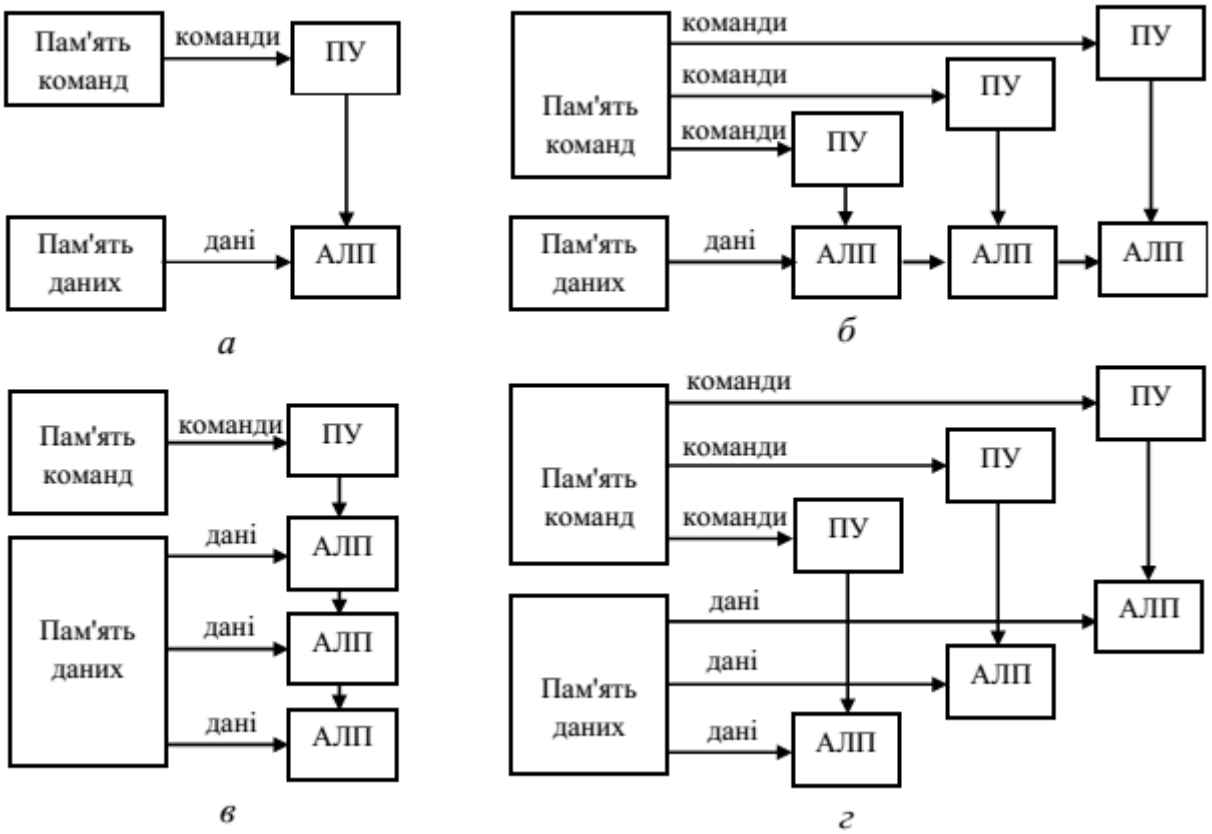


Рисунок 2.1 – Архітектура обчислювальних систем за Флінном: а – SISD; б –MISD; в – SIMD; г – MIMD

MIMD (Multiple Instruction Stream/Multiple Data Stream) – множинний потік команд і множинний потік даних. До цього класу відносяться системи з множиною пристроїв обробки команд, які об'єднані в єдиний комплекс і кожний працює з власним потоком команд. Цей клас надзвичайно широкий, бо включає в себе різного роду мультипроцесорні системи.

Запропонована схема класифікації аж до теперішнього часу є однією із найпопулярніших при початковій характеристиці певної обчислювальної системи. Наприклад, якщо відзначено, що комп'ютер належить до класу ОКМД (SIMD) або МКМД (MIMD), то відразу стає зрозумілим базовий принцип його роботи, і в деяких випадках цього буває достатньо.

Однак є явні **недоліки**. Зокрема, класифікація Флінна надто загальна, наприклад відносить усі паралельні комп'ютери, крім мультипроцесорних, до одного класу і не вказує ніякої відмінності між конвеєрними комп'ютерами і матрицею МП. Також, деякі обчислювальні системи (ОС) заслуговують більшої

уваги щодо архітектури, наприклад dataflow і векторно-конвеєрні машини, чітко не вписуються в класифікацію Флінна.

Інший недолік – це надмірна заповненість класу МКМД (MIMD). Зважаючи на викладене, відзначимо, що є потреба у таксономіях, які більш вибірково систематизують архітектури, що за таксономією Флінна потрапляють в один клас, але зовсім різні за кількістю процесорів, природою і топологією зв'язків між ними, за способом організації пам'яті і, звичайно ж, за технологією програмування, в тому числі паралельного програмування.

Розширення класифікації М. Флінна із врахуванням паралельних обчислювальних систем

Необхідно відзначити, що з розвитком паралельних обчислень та відповідних комп'ютерних технологій, була розширена відома класифікація Майкла Флінна (M.J. Flynn). Схеми SISD, SIMD, MISD, MIMD були розширені для паралельних обчислень до SPMD (Single-Program / Multiple-Data – одна програма, кілька потоків даних) і MPMD (Multiple-Programs / Multiple-Data – множина програм, велика кількість потоків даних) відповідно.

Схема SPMD (SIMD) дозволяє декільком процесорам виконувати одну і ту ж інструкцію або програму за умови, що кожен процесор отримує доступ до різних даних.

Схема MPMD (MIMD) дозволяє працювати декільком процесорам, причому всі вони виконують різні програми або інструкції і користуються власними даними.

Таким чином, в одній схемі всі процесори виконують одну і ту ж програму або інструкцію, а в іншій всі процесори виконують різні програми або інструкції. Звичайно ж, можливі гібридні реалізації цих моделей, в яких процесори можуть бути розділені на групи, з яких одні утворюють SPMD-модель, а інші – MPMD-модель. При використанні схеми SPMD всі процесори просто виконують одні й ті ж операції, але з різними даними. Якщо ж застосовується схема MPMD, всі процесори виконують різні види робіт, і хоча при цьому всі вони разом намагаються вирішити одну задачу, кожному з них визначається свій аспект цієї задачі.

Сучасна класифікація паралельних обчислювальних систем.

Розглянемо класифікацію паралельних ОС з урахуванням новоявлених сучасних архітектур. В основу класифікації (рис. 2.2) покладемо чотири базових класи класифікації М. Флінна (SISD, SIMD, MISD, MIMD), які розбиваються на підкласи відповідно з доповненнями учених Ванга та Бріггса.

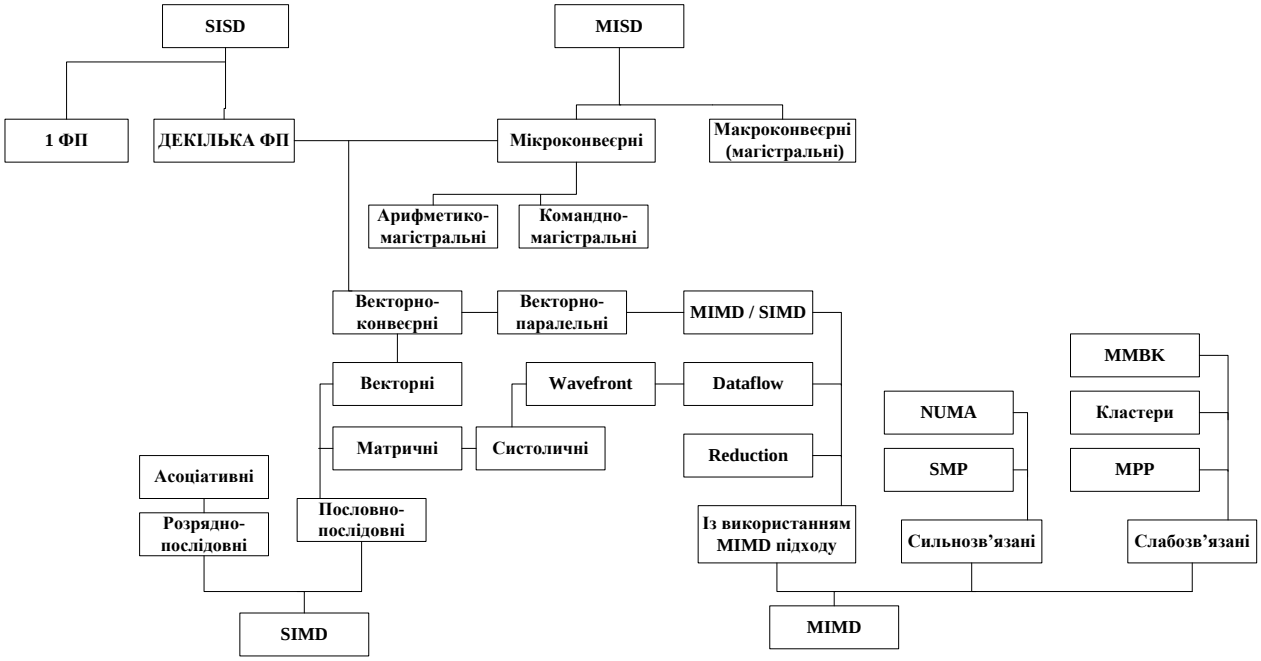


Рисунок 2.2 – Класифікація обчислювальних систем

Клас SISD поділяється на системи з одним функціональним пристроєм (ФП) і декількома ФП. Клас SIMD утворюють два підкласи – розрядно-послідовних і послівно-послідовних ОС. У класі MIMD виділимо сильно- і слабозв'язані ОС, а також ті ОС, що використовують ідеї MIMD, підкласи яких (MIMD/SIMD, Dataflow, Reduction, Wavefront) утворюються відповідно до класифікації Дункана. Нарешті, до класу MISD віднесемо комп'ютерні засоби та ОС, що використовують ідею конвеєрного оброблення. Паралелізм на рівні операторів і команд реалізується в мікроконвеєрних системах. При цьому системи, здатні розділяти за рівнями безпосереднє виконання однієї команди, утворюють підклас арифметико-магістральних систем. А системи, які конвеєризують всі етапи виконання команди (вибірка з пам'яті, дешифрація, вибірка операндів, виконання, запис результатів), визначають як командно-магістральні. Крім цього, виділено підклас макроконвеєрних (магістральних) ОС, здатних багаторазово вирішувати одну і ту ж обчислювальну задачу. При цьому, задача розбивається на послідовні частини, кожна з яких виконується на окремому процесорі (пристрої обробки). Вихідні дані однієї частини є вхідними – для наступної. За рахунок організації подібного макроконвеєра при багаторазовому повторенні обчислень з різними вхідними даними, отримують істотний виграш у продуктивності.

Переходячи до підкласів SIMD, помітно, що до розрядно-послідовних SIMD-систем можна віднести асоціативні ОС, а до послівно-послідовних – векторні та матричні. Систолічні масиви можна розглядати як окремий підклас матричних архітектур.

Серед сильнозв'язаних MIMD-архітектур виділяють симетричні мультипроцесори (SMP), що мають зосереджену загальну пам'ять, і архітектури з неоднорідним доступом до пам'яті (NUMA), в яких логічна спільна пам'ять фізично розподілена по вузлах системи. Слабозв'язані MIMD-системи представлені масивно-

паралельними (MPP), кластерними архітектурами, а також багатомашинними обчислювальними комплексами (БМОК). MPP-системи являють собою сукупність спеціалізованих обчислювальних модулів, об'єднаних високошвидкісними міжпроцесорними каналами зв'язку. Кластерні системи також являють собою приклад масового паралелізму, проте їх побудова здійснюється на основі стандартних промислових комплектуючих.

На рис. 2.2 також показано, що ряд ОС можуть бути одночасно віднесені до декількох класів або підкласів. Поєднання принципів мікроконвеєрного і векторного оброблення дає векторно-конвеєрні архітектури. Привносячи в них ідеї MIMD/SIMD, отримуємо векторно-паралельні (PVP) ОС. Машини *wavefront* є гібридними ОС, що керовані потоком даних (*dataflow*) і систолічними масивами.

Використовуються й інші класифікації архітектур, зокрема систематика Ф. Шара, структурна систематика Р. Хокні та К. Джессхоупа.

Структурна систематика Р. Хокні та К. Джессхоупа

На першому рівні всі обчислювальні системи поділяються за принципом множинності (кількості) на однокомп'ютерні та багатокомп'ютерні. Обчислювальні системи з одним комп'ютером, у свою чергу, поділяються на ОМ з одним конвеєрним МП та з багатьма МП.

Перші з них є традиційними послідовними комп'ютерами, а другі утворюють клас паралельних комп'ютерів, які поділяються на конвеєрні, неконвеєрні та мікропроцесорні матриці.

Прикладом однієї з перших неконвеєрних обчислювальних машин з паралелізмом є комп'ютер CDC – 6600, побудований на основі декількох скалярних процесорів.

Конвеєрні ЕОМ поділяються на такі, що виконують тільки скалярні команди, наприклад комп'ютери CDC – 7800, FPC AP-120B, і такі, що виконують векторні команди. Комп'ютери, що використовують векторні команди, поділяють, у свою чергу, на комп'ютери із спеціалізованим конвеєром, наприклад CRAY-1, та з універсальним конвеєром – комп'ютер CYBER 205.

Комп'ютери з класу машин з матрицею процесорів поділяють за зв'язаністю процесорів у матриці, розрядністю тощо. Першими машинами такого типу були ILLIAC-IV, BSP, STA-RAN, ICL DAP, OMEN та ін.

3 Рівні паралелізму

В умовах постійно зростаючих вимог до продуктивності обчислювальної техніки все більш явними стають обмеження класичної фон-нейманівської архітектури. Подальший розвиток обчислювальної техніки зв'язаний з переходом до паралельних обчислень як у рамках однієї обчислювальної машини, так і шляхом створення багатопроцесорних систем і мереж, які об'єднують велику кількість окремих процесорів або окремих обчислювальних машин. Для такого підходу замість терміну «обчислювальна машина» більш

підходить термін «обчислювальна система» (ОС). Головною особливістю такої системи є наявність у ній засобів, які реалізують паралельну обробку.

Паралельна обробка інформації являє собою одночасне рішення двох або більшої кількості частин однієї й тієї ж програми двома чи більшою кількістю ЕОМ (процесорними елементами) обчислювальної системи. Реалізують три основних способи організації паралельної обробки:

1) суміщення у часі різноманітних етапів різних задач – це мульти-програмна обробка, яка широко використовується як у однопроцесорних ЕОМ, так і в складних ОС;

2) одночасне розв'язання різноманітних задач або частин однієї задачі (можливо тільки за наявності декількох обробляючих пристроїв);

3) конвеєрна обробка інформації.

Перші два способи використовують особливості паралельних задач або потоків задач, що дозволяє здійснювати той або інший паралелізм. Перший тип паралелізму – це *природний паралелізм незалежних задач*, який полягає у тому, що в систему поступає безперервний потік не зв'язаних між собою задач. У цьому випадку розв'язання будь-якої задачі не залежить від результатів розв'язання інших задач, що дозволяє підвищити продуктивність ОС у разівикористання декількох обробляючих пристроїв.

Одним з найбільш розповсюджених типів паралелізму є *паралелізм незалежних гілок*. Суть його полягає у виділенні окремих незалежних частин великої задачі (гілок програми), які можуть виконуватись паралельно окремими обробляючими пристроями незалежно один від одного. Причому обробляючі пристрої ОС функціонують в однопрограмному режимі паралельної обробки інформації. Двома незалежними гілками програми визнаються гілки програми, що відповідають таким умовам:

- ні одна з вхідних для гілки програми величин не є вихідною величиною іншої програми (відсутність функціональних зв'язків);
- для двох гілок програми не повинен здійснюватись запис у одні й ті ж комірки пам'яті (відсутність зв'язку по оперативній пам'яті);
- умови виконання однієї гілки не залежать від результатів, що отримані під час виконання іншої гілки (незалежність по управлінню);
- обидві гілки повинні виконуватись у різних блоках програми (програмна незалежність).

Виділення незалежних гілок широко використовується під час паралельної обробки в задачах матричної алгебри, лінійного програмування, спектральної обробки сигналів, прямого та зворотного перетворення Фур'є та ін.

Методи та засоби реалізації паралелізму залежать від того, на якому рівні він повинен забезпечуватись. Зазвичай розрізняють такі *рівні паралелізму*:

- ***рівень завдань*** **–** декілька незалежних завдань одночасно виконуються на різних процесорах, які практично не взаємодіють один з одним. Цей рівень реалізується в ОС з множиною процесорів у багатозадачному режимі.
- ***рівень програм*** – частини однієї задачі виконуються множиною процесорів. Даний рівень досягається на паралельних ОС.

- ***рівень команд*** – виконання команди розділяється на фази, а фази декількох послідовних команд можуть бути перекриті за рахунок конвеєризації. Даний рівень використовується в ОС з одним процесором.
- ***рівень бітів (арифметичний рівень)*** – біти слова обробляються один за одним. Цей процес має назву *біт-послідовна операція*. Якщо біти слова обробляються одночасно, кажуть про *біт-паралельну операцію*. Даний рівень реалізується в звичайних і суперскалярних процесорах.

Паралелізм рівня завдання можливий між незалежними завданнями або їх фазами. Основним засобом реалізації паралелізму на рівні завдань є багатопроцесорні і багатомашинні обчислювальні системи, в яких завдання розподіляються за окремими процесорами або машинами. Однак, якщо кожне завдання трактувати як сукупність незалежних задач, реалізація даного рівня можлива і в рамках однопроцесорної ОС. У цьому випадку декілька завдань можуть одночасно знаходитись в основній пам'яті ОС, за умови, що в кожний момент виконується тільки одне з них. Коли завдання, яке виконується, потребує введення/виведення, ця операція запускається, а до її завершення інші ресурси ОС передаються другому завданню. По завершенні введення/виведення ресурси ОС повертаються до завдання, яке ініціювало цю операцію. В цьому випадку паралелізм забезпечується за рахунок того, що центральний процесор і система вводу/виводу функціонують одночасно і обслуговують різні завдання.

Паралелізм виникає також, коли у незалежних завдань, які виконуються в ОС, є декілька фаз, наприклад обчислення, запис у графічний буфер, системні виклики. За те, як різні завдання впорядковуються і витрачають загальні ресурси відповідає операційна система.

Паралелізм рівня програм. Про паралелізм на рівні програми можна говорити у двох випадках. По перше, коли в програмі можна виділити незалежні ділянки, які допустимо виконувати паралельно. Другий тип паралелізму програм можливий у межах окремого програмного циклу, якщо в ньому окремі ітерації не залежать один від одного. Програмний паралелізм можна реалізувати за рахунок великої кількості процесорів або множини функціональних блоків.

Загальна форма паралелізму на рівні програм організується розбиттям даних, які програмуються на підмножини. Цей розподіл називають *декомпозицією області* (domain decomposition), а паралелізм, який при цьому виникає, має назву *паралелізму даних*. Підмножини даних призначаються різним обчислювальним процесам, і цей процес має назву *розподілення даних* (data distribution). Процесори виділяються певним процесам або за ініціативою програми, або в процесі роботи операційною системою. На кожному процесорі може виконуватись більше одного процесу.

Паралелізм рівня команд. Паралелізм на рівні команд має місце, коли обробка декількох команд або виконання різних етапів однієї і тієї ж команди може перекриватись у часі. Розробники обчислювальної техніки ще здавна зверталися до методів, відомих під загальною назвою «поєднання операцій», при якому апаратура ОМ у будь-який момент часу виконує одночасно більше

однієї операції. Цей загальний принцип включає два поняття: *паралелізм* і *конвеєризацію*. Хоча у них багато загального і їх часто важко розрізняти на практиці, ці терміни відображають два принципово різних підходи.

У першому варіанті поєднання операцій досягається за рахунок того, що в складі обчислювальної системи окремі пристрої присутні в декількох копіях. Так, до складу процесора може входити декілька АЛП, і висока продуктивність забезпечується за рахунок одночасної роботи всіх цих АЛП.

Під час організації паралельного обчислювального процесу виникає задача вибору побудови розкладу.

Розклад паралельних обчислювальних процесів визначає порядок виконання програми в ОС, включаючи розподіл частин програми по обробляючих пристроях (процесорах, ОМ), і служить основою алгоритмів планувальника операційної системи та різноманітних управляючих програм. Як критерії оптимальних розкладів для паралельної програми можна назвати:

- мінімізацію часу виконання програми;
- мінімізацію кількості потрібних пристроїв обробки;
- мінімізацію середнього часу закінчення виконання завдань;
- максимізацію завантаження пристроїв ОС;
- мінімізацію часу простоювання пристроїв.

Найчастіше використовують перший критерій.

4 Комбінування паралельних і розподілених обчислень

Як уже зазначалось, при паралельних обчисленнях відразу декілька інструкцій можуть виконуватися в один і той же момент часу. Одна інструкція розбивається на кілька дрібних частин, які будуть виконуватися одночасно (їх можуть бути сотні чи навіть тисячі). Таким чином, *у паралельних обчисленнях послідовність і місце розташування складових програмного забезпечення не завжди передбачувані.* Декілька завдань можуть одночасно почати виконуватися на будь-якому процесорі без гарантії того, що завдання закріплені за певними процесорами, або ж що певна задача завершиться першою, або всі задачі завершаться у визначеному порядку.

Крім паралельного виконання задач, тут можливе паралельне виконання частин однієї задачі (підзадач). У деяких конфігураціях не виключена можливість виконання підзадач на різних процесорах або, навіть, різних КС. На рис. 2.3 зображено три рівні паралелізму, які можуть бути присутні в одній комп'ютерній програмі.

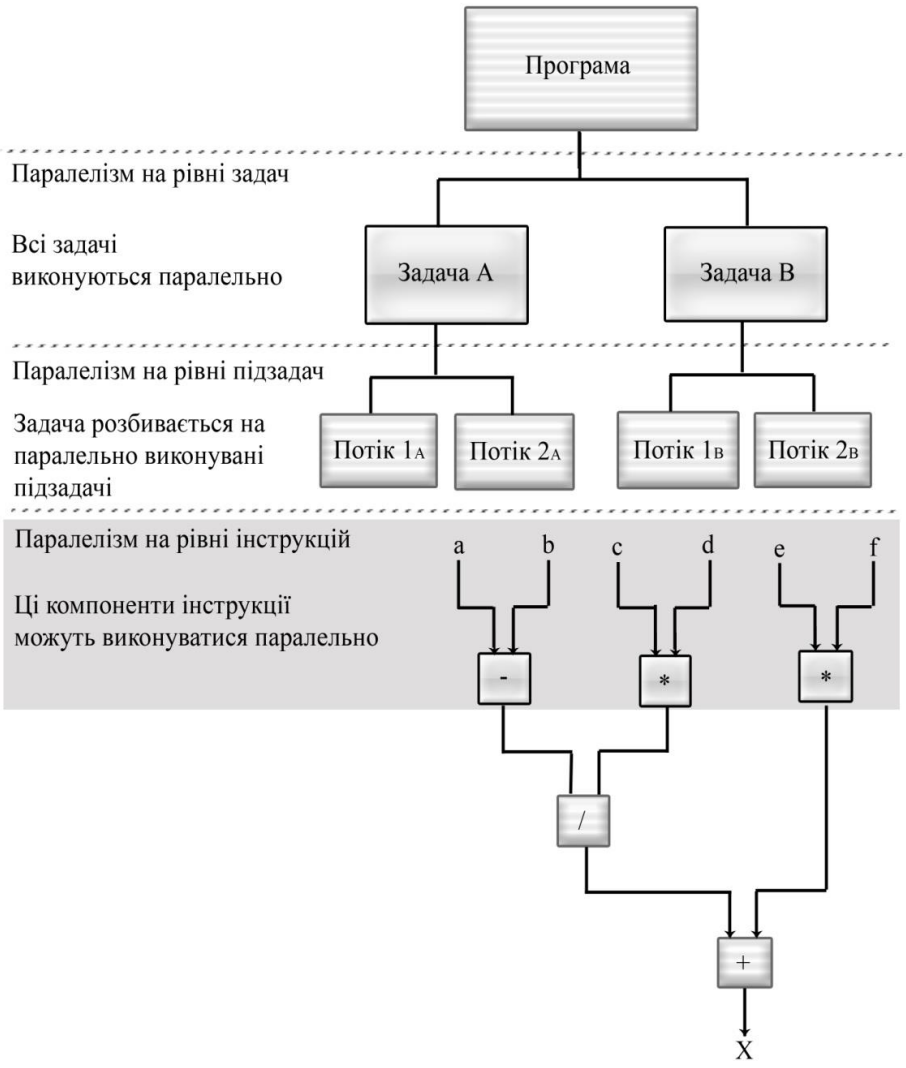
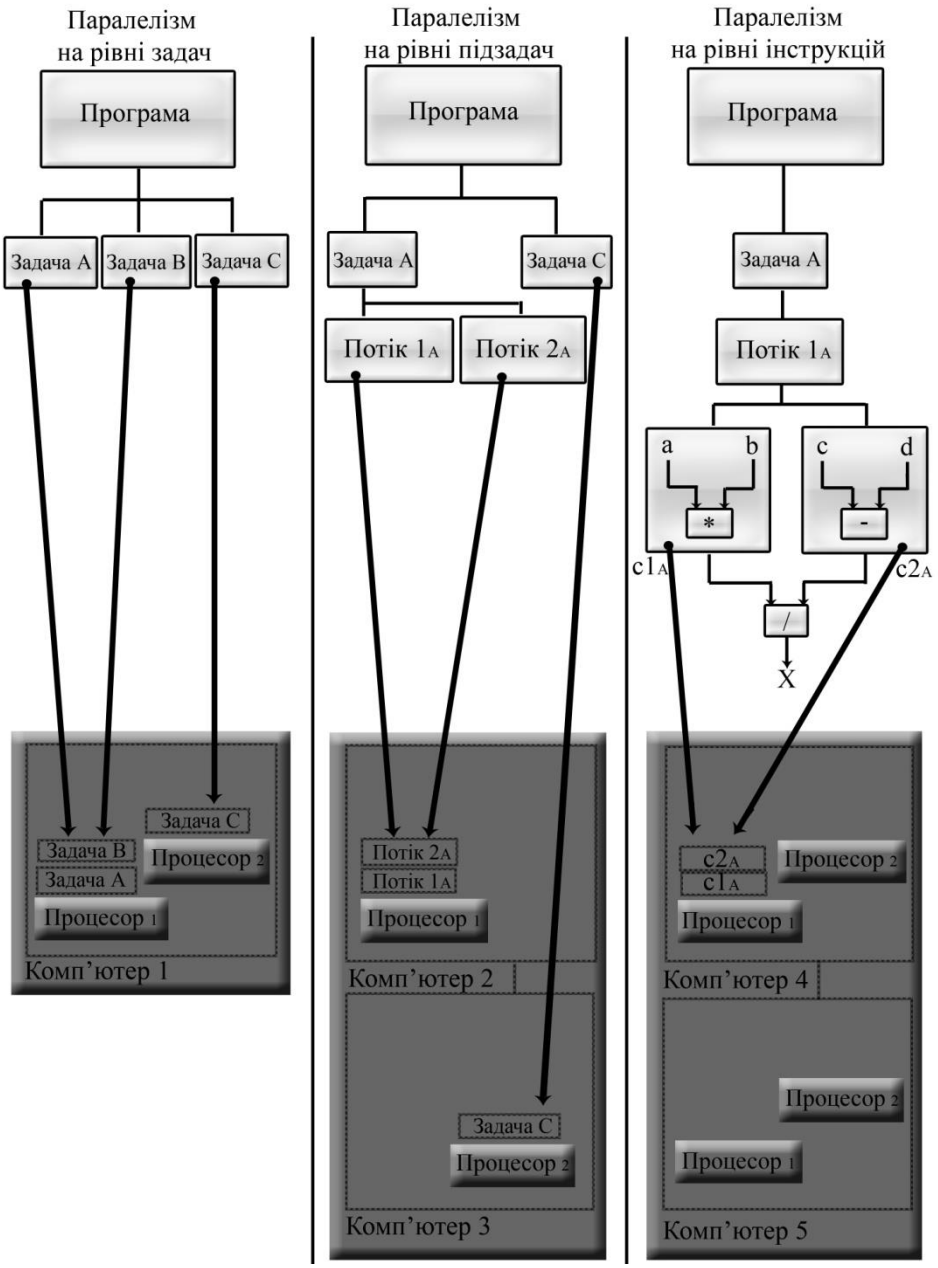


Рисунок 2.3 – Рівні паралелізму при організації обчислювального процесу

Модель програми (рис. 2.3) відображає кардинальну зміну парадигми обчислень. Тут відображено три рівні паралелізму та їх розподіл на декілька процесорів. Поєднання цих трьох рівнів з базовими паралельними конфігураціями процесорів показано на рис. 2.4.



а) б) в)

Рисунок 2.4 – Рівні паралелізму із врахуванням базових паралельних конфігурацій процесорів, де а) – однопроцесорна КС; б) – PVM-середовище з декількома однопроцесорними КС; в) – PVM-середовище з декількома багатопроцесорними КС

Додатково необхідно враховувати те, що декілька задач може виконуватися на одному процесорі, навіть при наявності в КС декількох процесорів. Така ситуація створюється системними стратегіями планування. На тривалість виконання задач, підзадач та інструкцій впливають і обрані стратегії планування, і пріоритети процесів, потоків, і швидкодія пристроїв введення-виведення.

Також, потрібно враховувати різноманітність архітектур при переході від послідовної моделі програмування до паралельної (рис. 2.4). *Основна відмінність тут полягає в переході від строго впорядкованої послідовності задач до лише*

частково впорядкованої (або зовсім невпорядкованою) колекції задач. Тобто, паралелізм перетворює раніше відомі величини (порядок виконання, час виконання і місце виконання) в невідомі. Будь-яка комбінація цих невідомих величин є причиною трансформаційних змін значень програми, причому зазвичай непередбачуваним чином.

Зважаючи на вищенаведене, необхідно відзначити, що процес проектування паралельних і розподілених КС та їх програмно-апаратного забезпечення повинен містити три складові: декомпозицію, зв'язок та синхронізацію.

Декомпозиція являє собою процес розбиття задачі та її розв’язання на частини. Іноді частини групуються в логічні області (наприклад, пошук, сортування, обчислення, введення/виведення даних і т.д.). В інших випадках частини групуються за логічними ресурсами (наприклад, файл, зв'язок, база даних і т.д.). *Декомпозиція програмного рішення часто зводиться до декомпозиції робіт – WBS (Work Breakdown Structure), яка визначає, що повинні робити різні частини програмного забезпечення.* Однією з основних проблем паралельного програмування є ідентифікація природної декомпозиції робіт для програмного рішення. Варто зазначити, що не існує простого та однозначного підходу до ідентифікації WBS. Це складний процес переведення принципів, ідей, шаблонів, правил, алгоритмів або формул в набір інструкцій, що виконуються, і даних, що обробляються КС. Це, в основному, і розкриває природну декомпозицію робіт програмного рішення. Чим краще модель зрозуміла і розроблена, тим більш природною буде декомпозиція робіт.

Після декомпозиції програмного рішення на ряд паралельно виконуваних частин зазвичай виникає питання про зв'язок цих частин між собою. На цьому етапі розглядається комплекс проблемних питань щодо: реалізації зв'язків при розподілі підзадач/задач по різних процесах або різних КС; спільного використання загальної області пам'яті різними частинами програмного забезпечення; комунікації між КС та повідомлення про завершення виконання підзадачі/задачі; черговості виконання підзадач/задач; комунікації між КС та повідомлення про відмову виконання, тощо. Якщо ж окремим частинам програмного забезпечення не потрібно зв'язуватися між собою, отже, вони насправді не утворюють єдиний додаток.

Синхронізація актуалізується в процесі виконання декомпозиції робіт. Коли компоненти програмного забезпечення працюють у рамках однієї задачі, їх функціонування необхідно координувати. Певний компонент повинен мати можливість визначити, коли досягається розв’язання усієї задачі. Важливо також скоординувати порядок виконання компонентів. При цьому також виникає комплекс проблемних питань щодо: одночасності виконання задачі та визначення черговості і організації перебування процесорів в стані очікування; організації коректного доступу до спільних ресурсів та пріоритетів; організації подальшої роботи процесорів після виконання підзадачі/задачі або ж у випадку дострокового виконання, тощо. Таким чином, декомпозиція, зв'язок і синхронізація – це той мінімум питань, які необхідно вирішити, приступаючи до паралельного або розподіленого програмування.

5 Високопродуктивні обчислювальні системи з гібридною архітектурою

Методи та середовища організації високопродуктивних паралельних і розподілених обчислень

Message Passing Interface.

Message Passing Interface (MPI, інтерфейс передачі повідомлень) - програмний інтерфейс (API) для передачі інформації, який дозволяє обмінюватися повідомленнями між процесами, які виконують одну задачу.

MPI є найпоширенішим стандартом інтерфейсу обміну даними в паралельному програмуванні. MPI реалізований для великого числа комп'ютерних платформ, і використовується при розробці програм для кластерів і суперкомп'ютерів. Основним засобом комунікації між процесами в MPI є передача повідомлень один одному.

У моделі програмування, яку підтримує MPI, програма породжує кілька процесів, що взаємодіють між собою за допомогою звернень до підпрограм передачі і прийому повідомлень. Зазвичай, при ініціалізації MPI-програми створюється фіксований набір процесів, причому кожен процес виконується на своєму процесорі. У цих процесах можуть виконуватися різні програми, тому модель програмування MPI іноді називають MPMD-моделлю (Multiple Program Multiple Data - безліч програм безліч даних), на відміну від SPMD-моделі, де на кожному процесорі виконуються тільки однакові завдання.

Parallel Virtual Machine.

Parallel Virtual Machine (PVM) (дослівно віртуальна паралельна машина) - загальнодоступний програмний пакет, що дозволяє об'єднувати різнорідний набір комп'ютерів в загальний обчислювальний ресурс («віртуальну паралельну машину») і надає можливості управління процесами за допомогою механізму передачі повідомлень. Існують реалізації PVM для різноманітних платформ від лептопів до суперкомп'ютерів Cray.

PVM дозволяє користувачам використовувати існуючі апаратні засоби для вирішення проблеми при мінімальних витратах. Сотні сайтів по всьому світу використовують PVM для вирішення важливих наукових, промислових і медичних проблем, PVM також використовується як освітній інструмент для навчання паралельного програмування.

З десятками тисяч користувачів, PVM став де-факто стандартом для розподілених обчислень в усьому світі.

Організація розподілених обчислень.

Методи організації розподілених обчислень дозволяють скористатися перевагами ресурсів, розміщених у корпоративних, глобальних і локальних мережах. Розподілені обчислення зазвичай містять мережеві обчислення в певній формі. Це означає, що програмі, що виконується на одній комп’ютерній системі

(КС) в одній мережі, потрібно деякий апаратний або програмний ресурс, який належить іншій КС в тій же або віддаленій мережі. Тобто, *методи розподіленого програмування надають доступ до ресурсів, які географічно можуть перебувати на великій відстані один від одного.* Методи розподіленого програмування передбачають спільний доступ до дороговартісних програмних і апаратних ресурсів. Розподілені обчислення можна використовувати для створення певного рівня резервування обчислювальних засобів на випадок аварії.

Найпростішою та найбільш поширеною моделлю розподіленого оброблення даних є *модель типу "клієнт/сервер".* Зазвичай між клієнтом і сервером існує відношення типу "багато-до-одного", тобто, як правило, один сервер відповідає на запити багатьох клієнтів. Незважаючи на те що модель типу "клієнт/сервер" – найпоширеніша модель розподіленого програмування, все ж вона не єдина. Використовуються також *агенти* – раціональні компоненти програмного забезпечення, які характеризуються самонаведенням та автономністю, і можуть постійно перебувати в стані виконання. Агенти співпрацюють в межах груп для колективного виконання певних завдань. У такої моделі не існує конкретного клієнта або сервера. Це модель мережі з рівноправними вузлами (*peer-to-peer*), в якій всі компоненти мають однакові права.

Найбільш поширеними середовищами для паралельного та розподіленого програмування є кластери, SMP- та MPP-комп'ютери.

Кластери – це колекції, що складаються з декількох КС, об'єднаних мережею для створення єдиної логічної системи. З точки зору програми така група КС виглядає як одна віртуальна ОС.

Під MPP-конфігурацією (процесори з масовим паралелізмом) розуміється один комп'ютерний засіб, що містить сотні процесорів, а під SMP-конфігурацією (симетричний мультипроцесор) – єдина система, в якій тісно пов'язані процесори спільно використовують загальну пам'ять та інформаційний канал.

На сучасному етапі паралельне багатопроцесорне оброблення є одним з основних напрямків для забезпечення високошвидкісного оброблення інформації. У глобальному аспекті, технологія паралельних обчислень розвивається в двох напрямках:

- обчислювальні системи з масовим (переважно, грубозернистим) паралелізмом на базі традиційних процесорів з послідовною системою команд, RISC і сигнальних процесорів, тощо;
- нейрокомп’ютинг – нейроподібні обчислювальні системи з масовим дрібнозернистим паралелізмом.

В останнє десятиліття було досягнуто значних результатів зі створення обчислювальних систем із паралельною архітектурою першого типу. Разом з тим, відбувався бурхливий розвиток досліджень і розробок зі створення й застосування обчислювальних систем на базі нейроподібних технологій. Причиною підвищеної уваги до нейрокомп’ютингу є властиві йому унікальні потенційні можливості виконання інтелектуальних операцій, а також можливості істотного підвищення критерію продуктивність/вартість порівняно з архітектурами традиційних комп'ютерних засобів. Такі можливості пояснюються наявністю в певній мірі

функціональної та структурної подібності з біологічними прототипами – нейронними мережами головного мозку людини (природній паралелізм).

Основна мета технологій паралелізму – забезпечити умови, що дозволяють комп'ютерними засобами здійснити більший обсяг роботи за той же період часу.

6 Технологія GPGPU

Практично будь-яка сучасна платформа виконання програмного коду, будь то повноцінна операційна система або віртуальна машина (наприклад, JAVA машина або .NET framework), що підтримує мультизадачність, містить набір API, призначений для управління потоками і створення паралельних програм. Таким чином, є можливість організувати паралельні обчислення практично на будь-якій мові від Assembler до скриптових мов типу Perl. Ясно, що проектувати паралельні програми не завжди виправдане рішення з точки зору витрат часу і якості коду, так як на розробника часто лягає безліч специфічних рутинних завдань зі створення, управління, контролю і забезпечення синхронізації потоків виконання. Йдеться, безумовно, про рішення обчислювально складних задач, так як прикладні програми створюються головним чином, використовуючи API платформи. На сьогоднішній день є бібліотеки і мови паралельного програмування, що спрощує безліч проблем, надаючи користувачу механізми для організації паралельних обчислень. Серед них можна відзначити MPI, PVM, мови Cisal, NESL, ZPL, Java, а також розширення різноманітних мов програмування.

Останнім часом почали приділяти увагу концепції GPGPU (General-purpose graphics processing units) - технології використання графічного процесора відеокарти для загальних обчислень, які зазвичай виконує центральний процесор.

Необхідність переходу до процесорів загального призначення

Аналіз різних графічних задач показав, що суворе закріплення функцій за вершинними і піксельними процесорами неефективне. Дійсно, якщо тривимірні моделі мають насичену геометрію, то в основному використовуються вершинні процесори, а якщо модель насичена піксельними ефектами, то основна робота відводиться піксельним процесорам. Це, в основному, і послужило аргументацією для компанії Nvidia для переходу на процесори загального призначення. Останні здатні виконувати не тільки функції вершинних і піксельних процесорів, але і виконувати різні розрахункові роботи загального призначення.

Аналіз шейдерних програм, що застосовуються Nvidia та ATI, показав неефективність використання обчислювальних ресурсів при векторній архітектурі виконавчих блоків. Це призвело до розуміння необхідності переходу в уніфікованих процесорах до скалярним обчислень, доручивши роботу по перетворенню векторних операцій в скалярні самому GPU, що і було зроблено компанією Nvidia в графічному процесорі GeForce 8800.

Згідно архітектурі GeForce 8800 вхідні дані (input stream) процесором обробляються, а його вихід (output stream) йде на вхід іншого процесора для

подальшої обробки. Циклічна потокова обробка дозволяє, якщо необхідно, провести повторну обробку даних, що зустрічається досить часто в графічних побудовах, без повторного введення вихідних даних або для їх подальших перетворень.

Для 3D відеоприскорювачів кілька років тому з'явилися перші технології нерафічних розрахунків. Сучасні відеочіпи містять сотні математичних виконавчих блоків, і ця тотужність може використовуватися для значного прискорення безлічі обчислювально інтенсивних додатків. Нинішні покоління GPU мають досить гнучку архітектуру, що разом з високорівневими мовами програмування робить їх значно доступнішими для складних обчислень.

Використання суперкомп'ютерів для паралельних обчислень стає все більш дорогим задоволенням. Крім того, суперкомп'ютери вимагають постійної модернізації для підтримки ефективності обчислень. У зв'язку з цим відбувається поступове переведення наукових обчислень на інші платформи. Застосування GPU процесорів дозволяє підвищити ефективність обчислень, використовуючи спеціалізовані бібліотеки. Відеокарти досить дешеві, легко замінні і до певних меж їх кількість можна нарощувати без особливих труднощів.

Останнім часом, популярними у використанні були чотири платформи, які реально втілили концепцію GPGPU.

AMD FireStream - технологія GPGPU, яка дозволяє програмістам реалізовувати алгоритми, що виконуються на графічних процесорах прискорювачів ATI.

CUDA - технологія GPGPU, яка дозволяє реалізовувати на мові програмування C алгоритми, що виконуються на графічних процесорах прискорювачів GeForce восьмого покоління і старше (GeForce 8 Series, GeForce 9 Series, GeForce 200, 300, 400 Series), Nvidia Quarod і Nvidia Tesla компанії Nvidia. Технологія CUDA розроблена компанією Nvidia. CUDA ToolKit 3.0 (Nvidia) підтримує OpenCL.

DirectCompute - набір інтерфейсів програмування додатків (API) компанії Microsoft є частиною останніх версій DirectX. Він призначений для виконання обчислень загального призначення на графічних процесорах. Безумовно, він може використовуватися і в ігровій практиці. Підтримується компаніями AMD і Nvidia.

OpenCL є мовою програмування задач, пов'язаних з паралельними обчисленнями на різних графічних і центральних процесорах. Мова програмування базується на стандарті C99. OpenCL включає також інтерфейс програмування додатків і забезпечує паралелізм на рівні інструкцій і на рівні даних.

DirectCompute - інтерфейс програмування додатків (API), який входить до складу DirectX - набору API від Microsoft, який призначений для роботи на IBM PC-сумісних комп'ютерах під управлінням операційних систем сімейства Microsoft Windows.

DirectCompute, з'явившись в складі DirectX 11, по суті став першою технологією в складі DirectX, яка надала доступ до обчислень загального призначення на графічних процесорах.

Незважаючи на орієнтацію на нерафічні обчислення загального призначення, DirectCompute може використовуватися і в ігровій графіці, наприклад, при рендерінгу тіней, рендерінгу напівпрозорих поверхонь без попереднього

сортування, а також при обробці і фільтрації цифрових зображень, прорахунку алгоритмів ігрового штучного інтелекту і для інших завдань.

Слід зазначити, що безліч додатків по молекулярному моделювання добре пристосоване для розрахунків на відеочіпах. На сьогоднішній день GPU пропонують високу продуктивність. Так, наприклад, GPU (Evergreen) позиціонують продуктивність до 10^{12} оп/с. Тому розумне використання тандему (CPU, GPU) дозволить істотно прискорити обробку великих масивів даних.

Основні відмінності між GPU і CPU полягають в архітектурних рішеннях (потокова і топографічна) та організації пам'яті (ієрархічна і модель з максимальною пропускною здатністю).

Однак, моделі програмування GPU (CUDA і OpenCL), які на тепер широко використовуються є низькорівневими, що вимагає від програміста значних зусиль з проектування практично повного сценарію обробки даних, в тому числі в трансформації звичайних програм і управління ресурсами.

Типова програма обчислень на CUDA виконує такі дії:

- копіює необхідні дані з оперативної пам'яті CPU в оперативну пам'ять GPU;
- задає розмірність блоків обчислень, їх кількість та ініціалізує процес обчислень на GUDA;
- кожен потік, який з'явився копіює частину необхідних для виконання блоку даних в швидку пам'ять, що розділяється;
- виконує обчислення;
- копіює результати виконання в основну пам'ять GPU;
- копіює результати з пам'яті GPU в основну оперативну пам'ять комп'ютера.

Технологія загальних обчислень

В обчислювальній моделі CUDA графічний процесор GPU можна розглядати як співпроцесор до CPU з власною пам'яттю, на який передаються обчислення для паралельної обробки великого числа потоків. При цьому накладні витрати на управління паралельною обробкою на GPU мінімальні в порівнянні з аналогічною акцією на CPU. Ясно, що чим більше потоків буде оброблятися на графічних процесорах, тим ефективніше вони будуть використовуватися.

Обробка на GPU + CPU здійснюється за схемою, яка подана на рис. 2.5. При цьому CUDA надає розробнику ефективних програмних проектів ряд функцій, які можуть виконуватися тільки на CPU, так званий CUDA host API.

Система CUDA володіє можливістю автоматичного розбиття оброблюваної частини на потоки та управління ними. При цьому всі потоки організовуються в ієрархію: множина потоків (grid), блоки і окремі потоки. Ясно, що взаємодію між потоками краще не допускати, щоб вартість обробки була мінімальною. Але якщо така взаємодія необхідна, то в CUDA для цього є два механізми: колективна (shared) пам'ять; бар'єрна синхронізація.

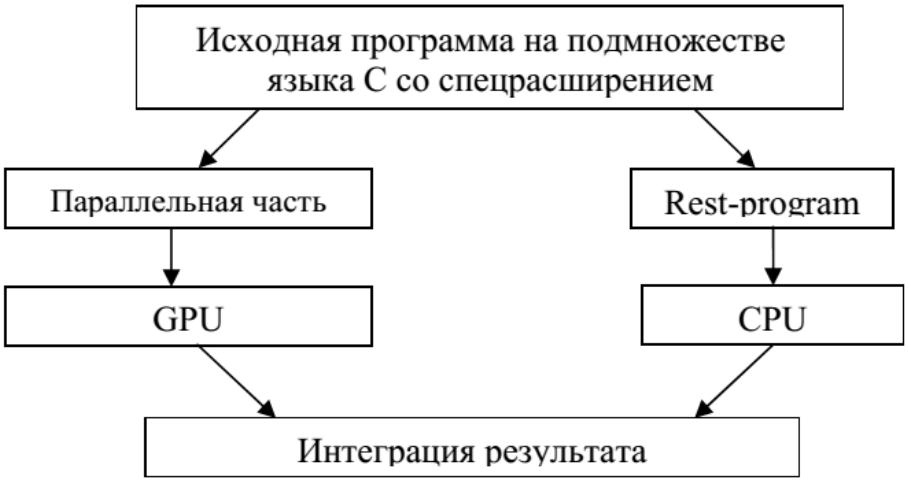


Рисунок 2.5 - Структура обробки програм на тандемі GPU + CPU

Оскільки, обробка потоків йде за технологією SIMD, то фактично виконується одна інструкція, але над різними даними. При цьому вихідна програма розробляється на «урізаному» С (немає операцій введення/ виведення, ряд функцій не підтримуються тощо), а відповідні файли мають розширення *.cu*. Якщо деякі функції можуть виконуватися як на CPU, так і на GPU, то відповідні специфікатори host і device можуть використовуватися разом.

Компілятор автоматично згенерує код для обох платформ.

Для кожного потоку будуть відомі: індекс потоку всередині блоку (threadldx, за замовчуванням вони передбачаються тривимірними) та індекс блоку всередині мереж (blockldx). Блок потоків виконується на мультипроцесорі пулами, кожен з яких включає, як правило, 32 потоки. При цьому всередині пулу одночасно може виконуватися тільки одна інструкція. Пул розбивається на частини, які кратні кількості процесорів в мультипроцесорі і виконуються послідовно. Обмін даними між завданнями всередині блоку здійснюється через загальну пам'ять, що розділяється.

Безумовно, можливі різні евристики по взаємодії процесів, але вони враховують особливості реальної апаратури і власні системні напрацювання. В якості останніх можна використовувати, наприклад, середовище реалізації Common Language Runtime (CLR) платформи .NET. Це дозволяє зробити реалізацію перетворення алгоритмів розв'язання задач на GPU незалежну від платформи і мови програмування. Щоб виконати паралельні фрагменти на GPU необхідно:

- завантажити модуль з необхідними функціями для відеокарти;
- виділити необхідний обсяг пам'яті на GPU;
- скопіювати дані з оперативної пам'яті в пам'ять відеокарти;
- проініціювати функції з завантаженого модуля, вказавши ступінь розпаралелювання;
- обробити дані;
- скопіювати результати з пам'яті GPU в пам'ять CPU.

Вимоги до задач для ефективного виконання на GPU

Аналіз технології обробки задач на відеокартах, архітектурних рішень CPU і GPU, а також проведення ряду експериментів на такому тандемі дозволяє стверджувати, що в першу чергу слід пропонувати задачі, які можна розпаралелити на сотні потоків. Особливо вагоме прискорення можна отримати, якщо одні й ті ж інструкції застосовуються до величезних масивів даних.

Наступною вимогою можна назвати відсутність взаємодій між потоками, які обробляються або «слабка» взаємодія. Серед інших властивостей задач (програм) для обробки на GPU можна назвати:

- мінімальна кількість складних для обробки операцій: ділення, піднесення у від’ємну степінь тощо;
- відсутність в алгоритмах множинного розгалуження;
- невеликий обсяг даних, переданих до відеокарти і від неї в оперативну пам'ять CPU.

Безумовно, іноді зазначені вимоги можна частково обійти за рахунок ретельного аналізу алгоритму розв'язання задачі і створення додаткових засобів, що знижують вплив зазначених та інших вимог на час повного циклу обчислення задач. Але все це треба робити надзвичайно коректно.