DeepL
Subscribe to DeepL Pro to translate larger documents
Visit www.DeepL.com/pro for more information.

Глава 10

Паралельні обчислення

Фон-нейманівська архітектура орієнтована на послідовне виконання команд програми. В умовах постійно зростаючих вимог до продуктивності обчислювальної техніки дедалі очевиднішими стають обмеження такого підходу. Подальший розвиток комп'ютерних засобів пов'язаний з переходом до паралельних обчислень, як у рамках однієї ОМ, так і шляхом створення багатопроцесорних систем і мереж, що об'єднують велику кількість окремих процесорів або окремих обчислювальних машин. Для такого підходу замість терміна "обчислювальна машина" більше підходить термін "обчислювальна система" (ОС). Відмінною особливістю обчислювальних систем є наявність у них засобів, що реалізують паралельне опрацювання, завдяки побудові паралельних гілок в обчисленнях, що не передбачалося класичною структурою ОМ.

Рівні паралелізму

Методи та засоби реалізації паралелізму залежать від того, на якому рівні він має забезпечуватися (рис. 10.1). Зазвичай розрізняють такі *рівні паралелізму*:

- *Мікрорівень*. Виконання команди поділяється на фази, а фази кількох сусідніх команд можуть бути перекриті за рахунок конвеєризації. Рівень досяжний на ВС з одним процесором.
- *Рівень команд.* Виражається в паралельному виконанні декількох команд і досягається за допомогою розміщення в процесорі відразу декількох конвеєрів. Реалізується в суперскалярних процесорах.
- *Рівень потоків*. Завдання розбиваються на частини, які можуть виконуватися паралельно (потоки). Цей рівень досягається на паралельних ВС.
- *Рівень завдань*. Кілька незалежних завдань одночасно виконуються на різних процесорах, практично не взаємодіючи одне з одним. Цей рівень реалізується на багатопроцесорних і багатомашинних ВС.

Рис. 10.1. Рівні паралелізму

До поняття рівня паралелізму тісно примикає поняття *гранулярності*. Це міра відношення обсягу обчислень, виконаних у паралельному завданні, до обсягу комунікацій (для обміну повідомленнями). Ступінь гранулярності варіюється від дрібнозернистої до грубозернистої. Визначимо поняття грубозернистого (coarse grained), середньозернистого (medium grained) і дрібнозернистого (fine grained) паралелізму.

Крупнозернистий паралелізм: кожне паралельне обчислення досить незалежно від інших, причому потрібен відносно рідкісний обмін інформацією між окремими обчисленнями. Одиницями розпаралелювання є великі і незалежні програми, що включають тисячі команд. Цей рівень паралелізму забезпечується операційною системою.

Середньозернистий паралелізм: одиницями розпаралелювання є викликані процедури, що включають в себе сотні команд. Зазвичай організовується як програмістом, так і компілятором.

Дрібнозернистий паралелізм: кожне паралельне обчислення досить мале й елементарне, складається з десятків команд. Зазвичай розпаралелюваними одиницями є елементи виразу або окремі ітерації циклу, що мають невеликі залежності за даними. Сам термін "дрібнозернистий паралелізм" говорить про простоту і швидкість будь-якої обчислювальної дії. Характерна особливість дрібнозернистого паралелізму полягає в приблизній рівності інтенсивності обчислень і обміну даними. Цей рівень паралелізму часто використовується розпаралелювальним (векторизуючим) компілятором.

Ефективне паралельне виконання вимагає вправного балансу між ступенем гранулярності програм і величиною комунікаційної затримки, що виникає між різними гранулами. Зокрема, якщо комунікаційна затримка мінімальна, то найкращу продуктивність обіцяє дрібноструктурне розбиття програми. Це той випадок, коли діє паралелізм даних. Якщо комунікаційна затримка велика, краще крупнозернисте розбиття програм.

Метрики паралельних обчислень

Метрики паралельних обчислень - це система показників, що дає змогу оцінити переваги, одержувані під час паралельного розв'язання задачі на n процесорах, порівняно з послідовним розв'язанням тієї самої задачі на єдиному процесорі. З іншого боку, вони дають змогу судити про обґрунтованість застосування даного числа процесорів для розв'язання конкретного завдання. Під *паралельними обчисленнями* розумітимемо послідовність кроків, де кожний крок складається з i операцій, що виконуються одночасно набором з i процесорів, які працюють паралельно.

Базисом для визначення згаданих метрик є такі характеристики обчислень:

- n - кількість процесорів, використовуваних для організації паралельних обчислень;
- $O(n)$ - обсяг обчислень, виражений через кількість операцій, що виконуються n процесорами під час розв'язання задачі;
- $T(n)$ - загальний час обчислень (розв'язання задачі) з використанням n процесорів.

Умовимося, що час змінюється дискретно, а за один квант часу процесор виконує будь-яку операцію. Унаслідок цього справедливі такі співвідношення для часу й обсягу обчислень: $T(1) = O(1)$, $T(n) \le O(n)$. Останнє співвідношення формулює твердження: *час обчислень можна скоротити за рахунок розподілу обсягу обчислень на кілька процесорів*.

Профіль паралелізму програми

Число процесорів, які паралельно виконують програму в кожен момент часу t, задає *ступінь паралелізму* $P(t)$ (Degree Of Parallelism). Графічне представлення параметра P у функції часу називають *профілем паралелізму програми*. Зміни кількості процесорів, що працюють паралельно (за час спостереження), залежать від багатьох чинників (алгоритму, доступних ресурсів, ступеня оптимізації, що забезпечується компілятором тощо). Типовий профіль паралелізму для алгоритму декомпозиції (divide-and-conquer algorithm) показано на рис. 10.2.

Припустимо, що система складається з n гомогенних (однорідних) процесорів. Продуктивність Δ одиночного процесора системи виразимо як кількість операцій за одиницю (квант) часу, не враховуючи витрат, пов'язаних зі зверненням до пам'яті та пересиланням даних. Якщо за спостережуваний період (певна кількість квантів часу) завантажені i процесорів, то $P = i$. Загальний обсяг обчислень $O(n)$ за період від стартового моменту $_{тн}$ до моменту завершення $_{тк}$ пропорційний площі під кривою профілю паралелізму:

$$O(n) = \Delta \sum_{i=1}^{n} {}_{iti},$$

де t_i інтервал часу (загальна кількість квантів часу), протягом якого $P = i$, а $\sum_{i=1}^{n} t_i = t_n - t_1$ - загальний час обчислень.

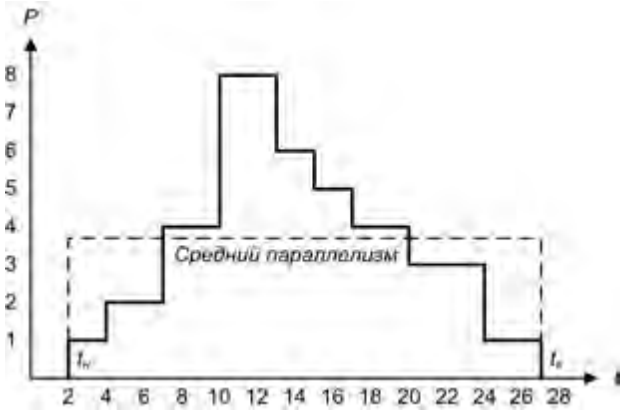

Рис. 10.2. Профіль паралелізму

Середній паралелізм A визначається як

$$= \frac{\sum_{i=1}^{n} iti}{n} A$$

$$\sum_{i=1} ti$$

Профіль паралелізму на малюнку за час спостереження ($_{тн}$, $_{тк}$) зростає від 1 до пікового значення n = 8, а потім спадає до 0. Середній паралелізм A = (1 × 5 + 2 × 3 + + 3 × 4 + 4 × 6 + 5 × 2 + 6 × 2 + 8 × 3) / (5 + 3 + 4 + 6 + 2 + 2 + 0 + 3) = 93/25 = 3,72.

Основні метрики

По суті, можна виділити чотири групи метрик.

Перша група характеризує швидкість обчислень. Ця група представлена парою метрик - *індексом паралелізму* і *прискоренням*.

Індекс паралелізму (Parallel Index) характеризує середню швидкість паралельних обчислень через кількість виконаних операцій:

$$PI(n) = \frac{O(n)}{T(n)}.$$

Прискорення (Speedup) за рахунок паралельного виконання програми слугує показником ефективної швидкості обчислень. Обчислюється прискорення як відношення часу, що витрачається на проведення обчислень на однопроцесорній ЗС (у варіанті найкращого послідовного алгоритму), до часу розв'язання тієї самої задачі на паралельній n-процесорній системі (при використанні найкращого паралельного алгоритму):

$$S(n) = \frac{T(1)}{T(n)}.$$

Зауваження щодо алгоритмів розв'язання задачі мають наголосити на тому факті, що для послідовної та паралельної реалізації найкращими можуть виявитися різні алгоритми, а під час оцінки прискорення необхідно виходити саме з найкращих алгоритмів.

Другу групу утворюють метрики *ефективність* і *утилізація*, що дають можливість судити про ефективність залучення до розв'язання завдання додаткових процесорів.

Ефективність (Efficiency) характеризує доцільність нарощування кількості процесорів через ту частку прискорення, досягнутого за рахунок паралельних обчислень, яка припадає на один процесор:

$$E(n) = \frac{S(n)}{n} = \frac{T(1)}{nT(n)}.$$

Утилізація (Utilization) враховує внесок кожного процесора під час паралельного обчислення, але у вигляді кількості операцій, виконаних процесором за одиницю часу.

$$U(n) = \frac{O(n)}{nT(n)}.$$

Третя група метрик, - *надмірність* і *стиснення*, - характеризує ефективність паралельних обчислень шляхом порівняння обсягу обчислень, виконаного під час паралельного та послідовного розв'язання завдання.

Надмірність (Redundancy) - це відношення обсягу паралельних обчислень до обсягу еквівалентних послідовних обчислень:

$$R(n) = \frac{O(n)}{O(1)}.$$

Важливість цієї метрики в тому, що вона виходить не з відносних показників прискорення та ефективності, отриманих із часу обчислень, а з абсолютних показників, що базуються на обсязі виконаної обчислювальної роботи. Надмірність відображає ступінь відповідності між програмним і апаратним паралелізмом.

Зазначимо, що утилізація може бути виражена через метрики надмірності та ефективності:

$$U(n) = \frac{O(n)}{nT(n)} = R(n)E(n).$$

Тут враховується співвідношення $T(1) = O(1)$.

Стиснення (Compression) обчислюється як величина, зворотна надмірності:

$$C(n) = \frac{O(1)}{O(n)}.$$

Нарешті, четверту групу утворює єдина метрика - *якість*, що об'єднує три розглянуті групи метрик.

Якість (Quality) визначається як:

$$Q(n) = \frac{T^3(1)}{nT^2(n)O(n)} = S(n)E(n)C(n).$$

Оскільки ця метрика пов'язує метрики прискорення, ефективності та стиснення, вона є більш об'єктивним показником поліпшення продуктивності за рахунок паралельних обчислень.

Для прикладу визначимо чисельні значення метрик стосовно задачі, використаної для ілюстрації поняття профілю паралелізму (рис. 10.2). Припускаючи, що найкращий алгоритм для послідовного і паралельного обчислення збігаються, маємо: $n = 8$; $T(1) = O(1) = O(1) = O(8) = 93$; $T(8) = 25$. Тоді:

$$PI(8) = \frac{O(8)}{T(8)} = \frac{93}{25} = 3,72; \qquad S(8) = \frac{T(1)}{T(8)} = \frac{93}{25} = 3,72;$$

$$E(8) = \frac{T(1)}{8T(8)} = \frac{93}{8 \times 25} = 0,465; \qquad U(8) = \frac{O(8)}{8T(8)} = \frac{93}{8 \times 25} = 0,465;$$

$$R(8) = \frac{O(8)}{O(1)} = \frac{93}{93} = 1; \qquad C(8) = \frac{O(1)}{O(8)} = \frac{93}{93} = 1;$$

$$Q(8) = S(8)E(8)C(8) = 3,72 \times 0,465 \times 1 = 1,73.$$

Насамкінець зазначимо, що для розглянутих метрик справедливі такі співвідношення:

$$1 \le S(n) \le PI(n) \le n; \quad 1 \le R(n) \le \frac{1}{E(n)} \le n; \quad \frac{1}{n} \le E(n) \le C(n) \le 1;$$

$$\frac{1}{n} \le E(n) \le U(n) \le 1; \quad Q(n) \le S(n) \le PI(n) \le n.$$

Закономірності паралельних обчислень

Купуючи для розв'язання своєї задачі паралельну обчислювальну систему, користувач розраховує на значне підвищення швидкості обчислень завдяки розподілу обчислювального навантаження по безлічі процесорів, що працюють паралельно. В ідеальному випадку система з n процесорів могла б прискорити обчислення в n разів. У реальності досягти такого показника не вдається через неможливість повного розпаралелювання жодного із завдань. Як правило, у кожній програмі є фрагмент коду, який принципово має виконуватися послідовно і тільки одним із процесорів. Це може бути частина програми, що відповідає за запуск завдання і розподіл розпаралеленого коду

по процесорах, або фрагмент програми, що забезпечує операції введення/виведення. Розпаралеленою може бути лише частина програми, що залишилася. Таким чином, програмний код розв'язуваної задачі складається з двох частин: послідовної і розпаралелюваної. Позначимо частку операцій, які мають виконуватися послідовно одним із процесорів, через f, де $0 \le f \le 1$ (тут частка розуміється не за числом рядків коду, а за числом реально виконуваних операцій). Частка, що припадає на розпаралелювану частину програми, складе $1 - f$. Крайні випадки в значеннях f відповідають повністю розпаралелюваним ($f = 0$) і повністю послідовним ($f = 1$) програмам. Цю ситуацію ілюструє рис. 10.3, у якому використано такі позначення:

- t_s - час обробки послідовної частини програми з використанням одного процесора;
- $t_p(1)$ - час опрацювання розпаралелюваної частини програми з використанням одного процесора;
- $t_p(n)$ - час обробки розпаралелюваної частини програми з використанням n процесорів.

Рис. 10.3 Ілюстрація розпаралелювання задачі в *n-процесорній* обчислювальній системі

Для розпаралелюваної частини задачі ідеальним був би варіант, коли паралельні гілки програми постійно завантажують усі процесори системи, причому так, щоб навантаження на кожний процесор було однаковим. На жаль, обидві ці умови на практиці важко реалізувати. Таким чином, орієнтуючись на паралельну ВС, необхідно чітко усвідомлювати, що домогтися збільшення продуктивності прямо пропорційного числу процесорів не вдасться, і, звісно, це не так,

постає питання про те, на яке реальне прискорення можна розраховувати. Відповідь залежить від того, яким чином користувач збирається використовувати обчислювальні потужності ВР, що зросли в результаті збільшення числа процесорів. Найхарактернішими є три варіанти:

1. Обсяг обчислень не змінюється, а головна мета розширення ВС - скоротити час обчислень. Досяжне в цьому випадку прискорення визначається законом Амдала.
2. Час обчислень із розширенням системи не змінюється, але при цьому збільшується обсяг розв'язуваної задачі. Мета такого підходу - за заданий час виконати максимальний обсяг обчислень. Цю ситуацію характеризує закон Густафсона.
3. Цей варіант схожий на попередній, але з однією умовою: збільшення обсягу розв'язуваної задачі обмежене ємністю доступної пам'яті. Прискорення в такому формулюванні визначає закон Сана і Ная.

Закон Амдала

Джин Амдал (Gene Amdahl) - один із розробників всесвітньо відомої системи IBM 360, у своїй праці [43], опублікованій у 1967 році, запропонував формулу, яка відображає залежність прискорення обчислень, що досягається на багатопроцесорній ЗС, як від кількості процесорів, так і від співвідношення між послідовною та розпаралелюваною частинами програми. Проблему розглядав Амдал, виходячи з положення, що обсяг розв'язуваної задачі (робоче навантаження - кількість виконуваних операцій) зі зміною кількості процесорів, які беруть участь у її розв'язуванні, залишається незмінним (рис. 10.4). Така постановка характерна для випадків, коли основною вимогою є швидкість обчислень, наприклад, для завдань, що виконуються в реальному часі. На малюнку $_{os}$ - це обсяг обчислень, зумовлених послідовною частиною програми; $_{op}(n)$ - обсяг обчислень, що припадає на розпаралелювану частину програми та розподіляється на n процесорів; $O(n)$ - загальний обсяг обчислень, який виконує n-процесорна система.

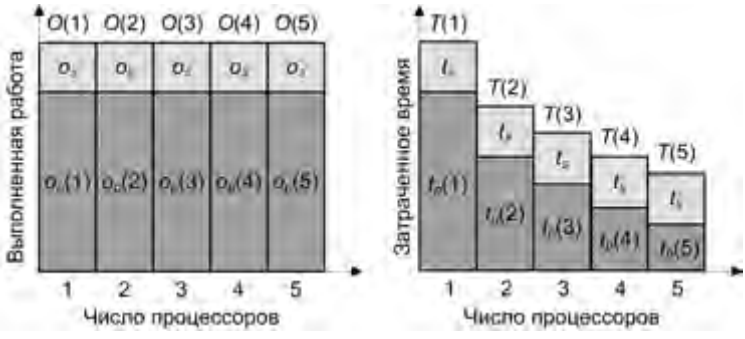

Рис. 10.4. Розподіл робочого навантаження і часу обчислень у формулюванні Амдала

З урахуванням "заморожування" об'єму задачі закон Амдала може бути представлений таким виразом:

$$S(n) = \frac{T(1)}{T(n)} = \frac{T(1)}{fT(1) + \frac{(1-f)T(1)}{n}} = \frac{1}{f + \frac{1-f}{n}}.$$

При безмежному збільшенні числа процесорів маємо:

$$\lim_{n\to\infty} S = \frac{1}{f}.$$

Це означає, що якщо в програмі 25% послідовних операцій (тобто f = = 0,25), то скільки б процесорів не використовували, прискорення роботи програми більш ніж у чотири рази ніяк не одержати, та й то і 4 - це теоретична верхня оцінка найкращого випадку, коли жодних інших негативних чинників немає.

Характер залежності прискорення від числа процесорів і частки послідовної частини програми показано на рис. 10.5.

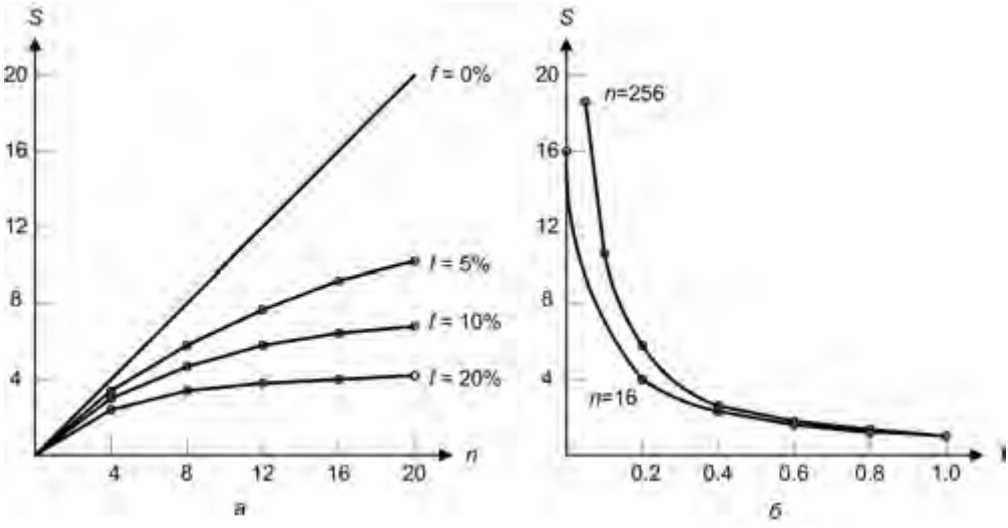

Рис. 10.5. Графіки залежності прискорення від: *а* - частки послідовних обчислень; *б* - числа процесорів

Слід зазначити, що розпаралелювання призводить до певних витрат, яких немає при послідовному виконанні програми. Як приклад таких витрат можна згадати додаткові операції, пов'язані з обміном інформацією між процесорами. З урахуванням часових витрат, зумовлених такими витратами (T_o), формула Амдала набуває вигляду:

$$S = \frac{T(1)}{T(n) + T_o} = \frac{T(1)}{fT(1) + \frac{(1-f)T(1)}{n} + T_o} = \frac{1}{f + \frac{1-f}{n} + \frac{T_o}{T(1)}}.$$

У цьому разі межа прискорення за нескінченного збільшення кількості процесів визначається виразом:

$$\lim_{n\to\infty} S(n) = \frac{1}{f}.$$

Отриманий вираз слугує аргументом проти обчислювальних систем з дуже великим числом процесорів, які розв'язують загальну задачу, оскільки в них витрати на обмін інформацією між процесорами, що реалізують паралельні частини цієї задачі, можуть виявитися вельми істотними.

Закон Густафсона

Відому частку оптимізму в оцінку, яку дає закон Амдала, вносять дослідження, проведені Джоном Густафсоном з NASA Ames Research [90]. Розв'язуючи на обчислювальній системі з 1024 процесорів три великі задачі, для яких частка послідовного коду f лежала в межах від 0,4 до 0,8%, він одержав прискорення порівняно з однопроцесорним варіантом, що дорівнюють відповідно 1021, 1020 і 1016. Відповідно до закону Амдала для цього числа процесорів і діапазону f, прискорення не повинно було перевищити величини порядку 201. Намагаючись пояснити це явище, Густафсон дійшов висновку, що причина криється у вихідній передумовою, що лежить в основі закону Амдала: збільшення кількості процесорів не супроводжується збільшенням обсягу розв'язуваної задачі. Реальна ж поведінка користувачів істотно відрізняється від такої гіпотези.

Зазвичай, отримуючи у своє розпорядження потужнішу систему, користувач не прагне скоротити час обчислень, а, зберігаючи його практично незмінним, намагається пропорційно до потужності ЗС збільшити обсяг розв'язуваної задачі, наприклад, щоб підвищити точність обчислень. І тут виявляється, що нарощування загального обсягу програми стосується головним чином розпаралелюваної частини програми. Це веде до скорочення величини f. Прикладом може слугувати розв'язання диференціального рівняння в частинних похідних. Якщо частка послідовного коду становить 10% для 1000 вузлових точок, то для 100 000 точок частка послідовного коду знизиться до 0,1%. Сказане ілюструє рис. 10.6.

Таким чином, підвищення прискорення зумовлене тим, що, залишаючись практично незмінною, послідовна частина в загальному обсязі збільшеної програми має вже меншу питому вагу. Щоб оцінити ступінь прискорення обчислень, коли обсяг останніх збільшується зі зростанням кількості процесорів у системі (за сталості загального часу обчислень), Густафсон рекомендує використовувати вираз, запропонований Е. Барсісом. Барсісом (Ed Barsis):

$$S(n) = f + (1 - f)n.$$

Цей вираз відомий як закон масштабованого прискорення або закон Густафсона (іноді його називають також законом Густафсона-Барсіса). Вираз констатує, що прискорення є лінійною функцією числа процесорів, якщо робоче навантаження масштабується так, щоб підтримувати незмінним час

обчислень. На закінчення ще раз зазначимо, що закон Густафсона не суперечить закону Амдала. Різниця полягає лише у формі використання додаткової потужності ВС, що виникає внаслідок збільшення числа процесорів.

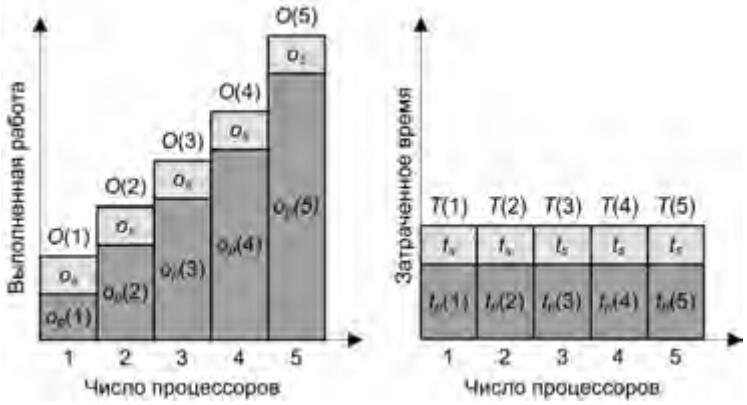

Рис. 10.6. Розподіл робочого навантаження і часу обчислень у формулюванні Густафсона

Закон Сана-Ная

У багатопроцесорній паралельній ВС кожен процесор зазвичай має незалежну локальну пам'ять порівняно невеликої ємності. Загальна пам'ять ВС утворюється об'єднанням локальної пам'яті кожного процесора ВС. Під час розв'язання задачу поділяють на підзадачі і розподіляють по безлічі процесорів. Підзадача розміщується в локальній пам'яті процесора. Як і в постановці Густафсона, збільшення кількості процесорів супроводжується зростанням розміру розв'язуваної задачі, але до межі, зумовленої ємністю доступної пам'яті. Іншими словами, обсяг завдання збільшується так, щоб кожна підзадача повністю займала локальну пам'ять процесора. Така постановка лежить в основі закону, сформульованого Ксіан-Хе Саном (Xian-He Sun) і Лайонелом Наєм (Lionel M. Ni), і має назву закону прискорення, обмеженого пам'яттю (рис. 10.7). На малюнку O* позначає робоче навантаження, обмежене доступною пам'яттю.

Нехай M - це ємність локальної пам'яті одного процесора. У цьому випадку сумарна пам'ять *n-процесорної* системи дорівнюватиме nM. У формулюванні проблеми з обмеженням, зумовленим пам'яттю, припускають, що пам'ять кожного процесора задіяна повністю, а робоче навантаження на один процесор дорівнює $O(1)$, де $O(1) = fO(1) + (1 - f)O(1)$. Покладемо, що під час використання всіх n процесорів розпаралелювана частина завдання може масштабуватися в $G(n)$ разів. У цьому разі масштабоване робоче навантаження, що масштабується, може бути описано виразом.

женням $O^* = fO + (1 - f\,)G(n)O$. Тут параметр $G(n)$ відображає зростання робочої сили.

чий навантаження зі збільшенням числа процесорів, а отже, і ємності пам'яті в n разів. У зазначених рамках прискорення описується виразом:

$$S(n) = \frac{f + (1 - f)G(n)}{f + (1 - f)\dfrac{G(n)}{n}}.$$

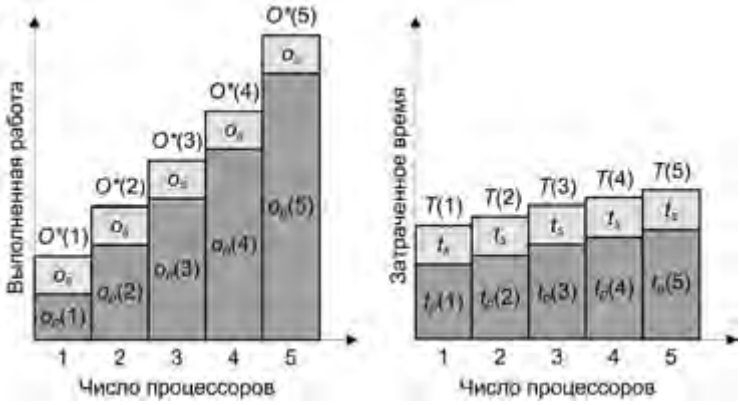

Рис. 10.7. Розподіл робочого навантаження і часу обчислень у постановці Сана і Ная

Неважко показати, що отриманий вираз являє собою узагальнення законів Амдала і Густафсона.

За $G(n) = 1$ розмір задачі фіксований, що відповідає постановці Амдала. У результаті отримуємо формулу Амдала:

$$S(n) = \frac{f + (1 - f)1}{f + (1 - f)\dfrac{1}{n}} = \frac{1}{f + \dfrac{1 - f}{n}}.$$

Варіант $G(n) = n$ відповідає випадку, коли зі збільшенням ємності пам'яті в n разів робоче навантаження також зростає в n разів. Це ідентично постановці Густаф- сона:

$$S(n) = \frac{f + (1 - f)n}{f + (1 - f)n} = f + (1 - f)n$$
$$f + (1 - f)n$$

У разі коли обчислювальне навантаження зростає швидше, ніж вимоги до пам'яті ($G(n) > n$), модель з обмеженням за пам'яттю дає більш оптимістичну оцінку прискорення.

Малюнок 1.8 ілюструє оцінки прискорення, отримані за кожною з трьох розглянутих моделей прискорення.

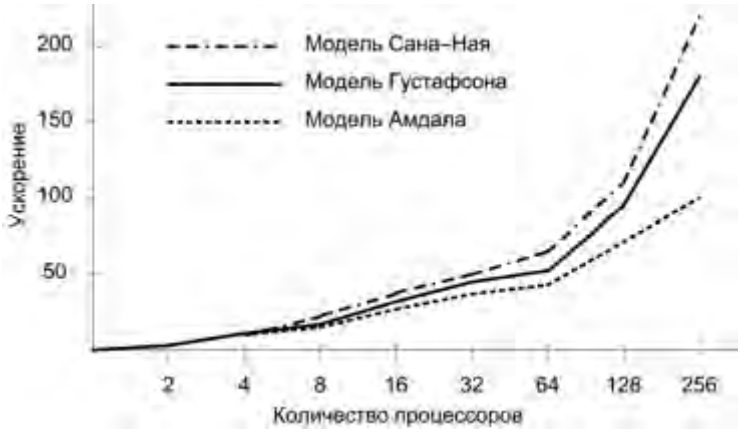

Рис. 10.8. Порівняння трьох моделей прискорення

Метрика Карпа-Флетта

У виразах, що представляють закони Амдала, Густафсона і Сана-Ная, крім числа процесорів, фігурує частка послідовних обчислень *f*. Зазвичай її визначають шляхом аналізу коду програми і з'ясування того, яка за обсягом обчислень частина програми не може бути розпаралелена. Водночас значення прискорення, одержувані на реальних системах, нижчі, ніж передбачають відповідні формули. Викликано це, головним чином, неврахуванням раніше згадуваних витрат на взаємодію між процесорами, що працюють паралельно. Для оцінки реальної можливості розпаралелювання конкретного коду в паралельній ЗС Алан Карп (Alan H. Karp) і Горас Флетт (Horace P. Flatt) запропонували використовувати еквівалент показника *f*, відомий як міра *Карпа-Флетта, що має* позначення *e*. Цей показник обчислюють виходячи з експериментально визначеного прискорення на реальній ПС за такою формулою:

$$e = \frac{\frac{1}{S(n)} - \frac{1}{n}}{1 - \frac{1}{n}}.$$

Що менше значення *e*, то краще може бути розпаралелений код. Для задач фіксованого розміру (як у постановці Амдала) ефективність паралельних обчислень зі збільшенням числа процесорів зазвичай зменшується. За допомогою показника *e*, отриманого на підставі експериментальних даних (метрики Карпа-Флетта), можна оцінити, чим саме зумовлено зниження ефективності - обмеженими можливостями розпаралелювання чи комунікаційними витратами паралельного обчислення (часом на обмін інформацією між паралельними гілками).

Класифікація паралельних обчислювальних систем

Навіть коротке перерахування типів сучасних паралельних обчислювальних систем (ПС) дає зрозуміти, що для орієнтації в цьому різноманітті необхідна чітка система класифікації. Від відповіді на головне питання - що закласти в основу класифікації - залежить, наскільки конкретна система класифікації допомагає розібратися з тим, що являє собою архітектура ЗС і наскільки успішно дана архітектура дає змогу розв'язувати певне коло завдань. Спроби систематизувати всю безліч архітектур паралельних обчислювальних систем робили досить давно і тривають донині, але до однозначних висновків поки що не привели. Вичерпний огляд наявних систем класифікації ВС наведено в [5].

Класифікація Флінна

Серед усіх розглянутих систем класифікації ВС найбільшого визнання набула класифікація, запропонована 1966 року М. Флінном [84, 85]. В її основу покладено поняття потоку, під яким розуміють послідовність елементів, команд або даних, оброблювану процесором. Залежно від кількості потоків команд і потоків даних Флінн виділяє чотири класи архітектур: SISD, MISD, SIMD, MIMD.

SISD

SISD (Single Instruction Stream/Single Data Stream) - одиночний потік команд і одиночний потік даних (рис. 10.9, *а*). Представниками цього класу є, насамперед, класичні фон-нейманівські ВМ, де є тільки один потік команд, команди обробляються послідовно і кожна команда ініціює одну операцію з одним потоком даних. Те, що для збільшення швидкості опрацювання команд і швидкості виконання арифметичних операцій може застосовуватися конвеєрне опрацювання, не має значення, тому в клас SISD одночасно потрапляють як ВМ CDC 6600 зі скалярними функціональними пристроями, так і CDC 7600 з конвеєрними. Деякі фахівці вважають, що до SISD-систем можна зарахувати і векторно-конвеєрні ВМ, якщо розглядати вектор-тор як неподільний елемент даних для відповідної команди.

MISD

MISD (Multiple Instruction Stream/Single Data Stream) - множинний потік команд і одиночний потік даних (рис. 10.9, *б*). З визначення випливає, що в архітектурі ВС присутня безліч процесорів, які обробляють один і той самий потік даних. Прикладом могла б слугувати ВС, на процесори якої подається спотворений сигнал, а кожен із процесорів обробляє цей сигнал за допомогою свого алгоритму фільтрації. Проте ні Флінн, ні інші фахівці в галузі архітектури комп'ютерів досі не зуміли представити переконливий приклад реально існуючої обчислювальної системи, побудованої за допомогою

на цьому принципі. Низка дослідників [92, 95, 156] відносять до цього класу конвеєрні системи, однак це не знайшло остаточного визнання. Звідси прийнято вважати, що поки цей клас порожній. Наявність порожнього класу не слід вважати недоліком класифікації Флінна. Такі класи, на думку деяких дослідників [73, 137], можуть стати надзвичайно корисними для розроблення принципово нових концепцій у теорії та практиці побудови обчислювальних систем.

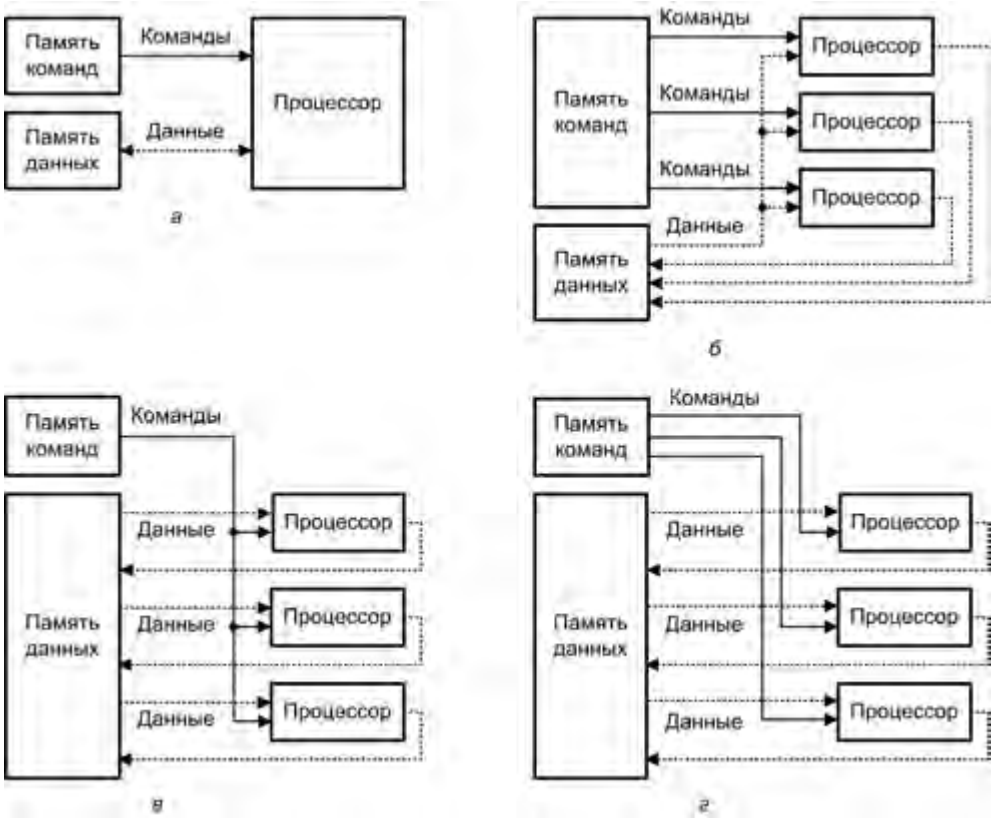

Рис. 10.9. Архітектура обчислювальних систем за Флінном: *а* - SISD; *б* - MISD; *в* - SIMD; *г* - MIMD

SIMD

SIMD (Single Instruction Stream/Multiple Data Stream) - одиночний потік команд і множинний потік даних (рис. 10.9, *в*). ВМ цієї архітектури дають змогу виконувати одну арифметичну операцію відразу над багатьма даними - елементами вектора. Безперечними представниками класу SIMD вважаються матриці процесорів, де єдиний керуючий пристрій контролює безліч процесорних елементів. Усі процесорні елементи отримують від пристрою керування однакову команду і виконують її над своїми локальними даними.

У принципі, до цього класу можна включити і векторно-конвеєрні ВС, якщо кожен елемент вектора розглядати як окремий елемент потоку даних.

MIMD

MIMD (Multiple Instruction Stream/Multiple Data Stream) - множинний потік команд і множинний потік даних (рис. 10.9, *г*). Клас передбачає наявність в обчислювальній системі безлічі пристроїв оброблення команд, об'єднаних у єдиний комплекс, які працюють кожне зі своїм потоком команд і даних. Клас MIMD надзвичайно широкий, оскільки містить у собі всілякі мультипроцесорні системи. Крім того, долучення до класу MIMD залежить від трактування. Так, раніше згадувані векторно-конвеєрні ВС можна цілком віднести і до класу MIMD, якщо конвеєрне опрацювання розглядати як виконання безлічі команд (операцій ступенів конвеєра) над множинним скалярним потоком.

Схема класифікації Флінна аж до теперішнього часу є найпоширенішою під час первісної оцінки тієї чи іншої ВС, оскільки дає змогу відразу оцінити базовий принцип роботи системи, чого часто буває достатньо. Однак у класифікації Флінна є й очевидні недоліки, наприклад нездатність однозначно віднести деякі архітектури до того чи іншого класу. Інша слабкість - це надмірна насиченість класу MIMD. Усе це породило численні спроби або модифікувати класифікацію Флінна, або запропонувати іншу систему класифікації.

Контрольні питання

1. Порівняйте схеми класифікації паралелізму за рівнем і гранулярністю. Які, на ваш погляд, переваги, недоліки та сфери застосування цих схем класифікації?
2. У чому полягають основні ідеї виділення чотирьох груп метрик паралельних обчислень?
3. Доведіть інтегральний характер метрики "якість".
4. Для заданої програми та конфігурації паралельної обчислювальної системи розрахуйте значення метрик паралельних обчислень.
5. Поясніть суть закону Амдала, наведіть приклади, що пояснюють його обмеження.
6. Яку проблему закону Амдала вирішує закон Густафсона? Як він це робить? Сформулюйте сфери застосування цих двох законів.
7. Чому закон Сана-Ная називають узагальненням законів Амдала і Густафсона? Доведіть це твердження.
8. Яке завдання вирішує метрика Карпа-Флетта і яким саме чином?
9. Укажіть переваги та недоліки схеми класифікації Флінна.

Глава 11

Пам'ять обчислювальних систем

В обчислювальних системах, що об'єднують безліч процесорів або машин, які працюють паралельно, завдання правильної організації пам'яті є одним з найважливіших. Різниця між швидкодією процесора і пам'яті завжди була каменем спотикання в однопроцесорних ВМ. Багатопроцесорність ВМ призводить ще до однієї проблеми - проблеми одночасного доступу до пам'яті з боку декількох процесорів.

Залежно від того, яким чином організовано пам'ять багатопроцесорних (багатомашинних) систем, розрізняють обчислювальні системи з парою пам'яті, яку можна розділити (shared memory), та обчислювальні системи з розподіленою пам'яттю (distributed memory).

У *системах із пам'яттю, що розділяється* (її часто називають також спільно використовуваною або загальною пам'яттю), пам'ять ВЗ розглядають як загальний ресурс, і кожен із процесорів має повний доступ до всього адресного простору. Системи з поділюваною пам'яттю називають *сильно пов'язаними* (closely coupled systems). Подібна побудова обчислювальних систем має місце як у класі SIMD, так і в класі MIMD. Іноді, щоб підкреслити цю обставину, вводять спеціальні підкласи, використовуючи для їхнього позначення абревіатури SM-SIMD (Shared Memory SIMD) і SM-MIMD (Shared Memory MIMD).

У варіанті *з розподіленою пам'яттю* кожному з процесорів надається власна пам'ять. Процесори об'єднуються в мережу і можуть за необхідності обмінюватися даними, що зберігаються в їхній пам'яті, передаючи один одному так звані *повідомлення*. Такий вид ВС називають *слабо пов'язаними* (loosely coupled systems). Слабко зв'язані системи також трапляються як у класі SIMD, так і в класі MIMD, і іноді, щоб підкреслити цю особливість, вводять підкласи DM-SIMD (Distributed Memory SIMD) і DM-MIMD (Distributed Memory MIMD).

Іноді обчислювальні системи з пам'яттю, що розділяється, називають *мультипроцесорами*, а системи з розподіленою пам'яттю - *мультикомп'ютерами*.

Різниця між розподіленою та розподіленою пам'яттю полягає у структурі віртуальної пам'яті, тобто в тому, як пам'ять має вигляд з боку процесора. Фізично майже кожна система пам'яті розділена на автономні компоненти,