<u>Тема №2</u> " Структури паралельних та розподілених КС. Паралельні алгоритми. Представлення, побудова та аналіз. Методи оцінки продуктивності паралельних алгоритмів і систем "

Питання:

1. **Основні положення паралельних та розподілених комп'ютерних систем**
2. **Шинні мережі**
3. **Мережі з комутаторами**
4. **Структури, що забезпечують зв'язок типу «пункт-пункт»**
5. **Методи комутацій**
6. **Загальні зауваження стосовно оцінки продуктивності паралельних систем**
7. **Фактори, що необхідно враховувати при оцінці продуктивності**
8. **Методи оцінки продуктивності паралельних систем**
9. **Характеристики продуктивності паралельних алгоритмів**
10. **Порівняння MIMD і SIMD структур за продуктивністю**

Вправи і завдання до лекції №2

1. Основні положення паралельних та розподілених комп'ютерних систем

Кожна паралельна система/процесор складається з ряду процесорів і модулів пам'яті, які зв'язані через відповідні комутаційні структури. Всі концепції комунікацій, такі як "schared memory" (загальна пам'ять, реальна або віртуальна), або обмін повідомленнями реалізуються в паралельних системах деякими наявними структурами. Враховуючи задачі обчислювальної системи, її структура зв'язку має відповідати *різним критеріям*, насамперед, *забезпечувати* по можливості *високу коннективність* (гарантувати зв'язок між двома будь-якими процесорами або модулями пам'яті без потреби переходу через проміжні пункти комутації). Окрім цього, має забезпечуватись *найбільша кількість одночасних зв'язків*, - щоб комутаційна мережа не обмежувала продуктивність вузлів паралельної обробки інформації. Проте об'єктивно існують різноманітні обмеження на реалізацію цих критеріїв: *кількість ліній зв'язку на один процесор* не може збільшуватися безмежно, має також фізичні обмеження і *ширина частотної смуги пропускання* (швидкість передачі) комутаційної мережі.

Для паралельної системи, яка складається з процесорних елементів (ПЕ) можна визначити такі види витрат:

- *кількість зв'язків на один ПЕ (витрати на виготовлення системи);*
- *дистанція між ПЕ (витрати під час експлуатації).*

Тобто, кількість зв'язків на один процесор і дистанція, тобто найкоротший шлях між двома заданими ПЕ, мають бути якомога меншими. Структура зв'язку має бути розширюваною за масштабом, тобто малі мережі зв'язку мають збільшуватися завдяки доповненням.

Структури зв'язку поділяються на три великих класи:

- шинні сітки;
- сітки з комутаторами;
- структури, що забезпечують зв'язок типу "пункт-пункт".

Причому, для всіх цих структур основною характеристикою мережі залишається *пропускна здатність,* одиницею вимірювання якої є *біт/с*.

Наведемо основні *визначення*, які відносяться до структури зв'язків.

Топологія – організація внутрішніх комунікацій обчислювальної системи.

Два вузли є *сусідніми*, якщо між ними є прямі з'єднання.

Порядком вузла називається кількість його сусідів (позначається V – кількість зв'язків).

Комунікаційним діаметром мережі є мінімальний шлях між двома сусідніми вузлами (позначається A – віддаль).

Масштабованість характеризує зростання складності з'єднань при добавленні в конфігурацію нових вузлів. (При довільній масштабованості системи, її складність при нарощуванні буде незначно змінюватись, незмінним буде діаметр мережі)

Є два типи технології комутації мереж – *статичні* і *динамічні*.

2. Шинні мережі

В паралельних структурах засобами шини можна з'єднати між собою процесори або модулі пам'яті (рис.5.1).

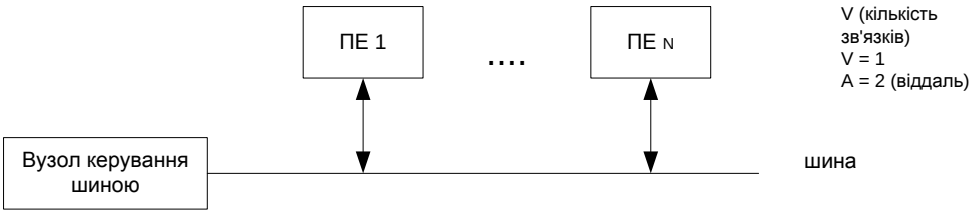


Рис. 2.1. Шинна система

Кількість зв'язків на один ПЕ є оптимальною, віддаль між ПЕ є незмінною і дорівнює 2 (кожний зв'язок йде від ПЕ до шини і назад). Це також практично оптимально число. *Недоліки шинної організації* зумовлені тим, що в кожний момент може відбуватись тільки одне з'єднання. Паралельне читання за однією і тією адресою вузла пам'яті можливе, однак запис - ні. Процесори не можуть незалежно обмінюватися даними; паралельна обробка при обміні даними між процесорами неможлива.

Система керування шиною має гарантувати послідовне упорядкування одночасних запитів на послуги шини.

Якщо система може розширюватися на багато процесорів, то ширина частотної смуги пропускання шини залишається незмінною. Цей важливий недолік визначає масштабні спроможності шинних структур. Тому шинні системи як засоби зв'язку в паралельних ЕОМ, в яких кількість процесорів перевищує десять, застосовувати не рекомендується.

3. Мережі з комутаторами

Мережі з комутаторами - це динамічні структури зв'язку, в яких за допомогою ліній керування реалізовані різні варіанти спілкування процесорів перед початком виконання паралельної програми. Розглянемо динамічні сітки типів: *розподілювача перехресних шин, дельта-сітки, мережі зв'язку Клоса та сітки типу Fat-Tree.* На рис.2.2 наведена схема *розподілювача перехресних шин.*

Кожний ПЕ має *n-1* пункт комутації, оскільки діагональні пункти не потребують контакту. Загалом сітка має *n(n-1)* пунктів комутації і дає можливість встановити будь-яку кількість з'єднань між всіма ПЕ, тобто без всяких колізій може відбуватись повністю паралельний обмін даними. Суттєвим недоліком цього комутатора є витрати на його реалізацію , що становлять (n^2-n) пунктів з'єднань для *n* ПЕ. Практично така кількість пунктів може бути реалізована лише для невеликої кількості процесорів. Вже для 100 ПЕ потрібно 9900 перемикань, щоб забезпечити роботу всіх пунктів з'єднань.

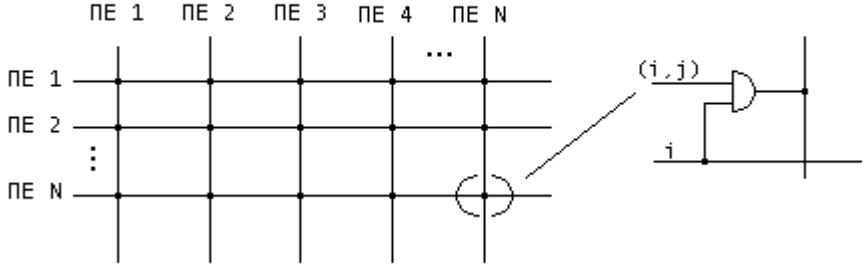


Рис. 2.2. Розподілювач перехресних шин

Дельта - сітки

Щоб зменшити витрати, що виникають під час побудови розподілювачів перехресних шин, використовуються дельта-сітки (див. рис.2.3).

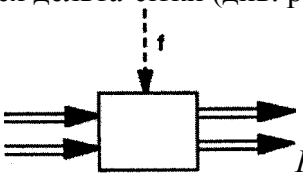


Рис. 2.3. Дельта-мережі комутації

В найпростішому випадку (рис.2.3) дві лінії даних можуть перемикатись навхрест за допомогою єдиної лінії керування або на пряму передачу з входу на вихід.

На рис.2.4 показано, як проходять сигнали всередині "чорного ящика" деякої дельта-сітки. Якщо керуючий сигнал дорівнює “нулю”, то обидві лінії даних (або пучки ліній) приєднуються прямо до вихідних ліній. У випадку, коли керуючий сигнал дорівнює “одиниці”, лінії даних перемикаються на виході навхрест.

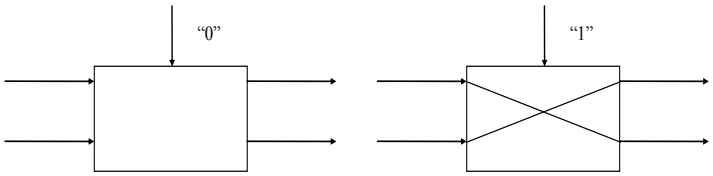


Рис. 2.4. Перемикач дельта-мережі

3 таких простих базових елементів можна будувати більш об’ємні сітки.

На рис.2.5 наведено триступеневу дельта-сітку, яка з’єднує 8 входів з 8 виходами. Кожний елемент сітки відповідає базисному елементу, що показаний на рис.2.3.

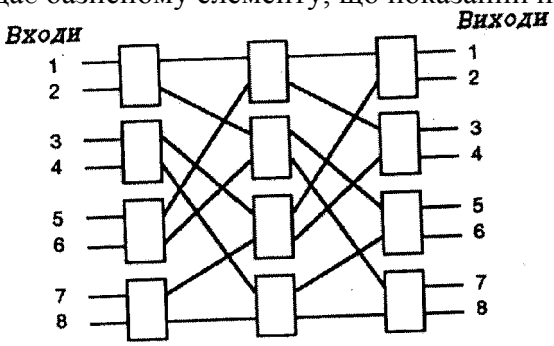


Рис. 2.2. Дельта-сітка розміром 8x8

Перевагою дельта-сіток над розподілювачами перехресних шин є менша кількість дельта-перемикачів, $\frac{n}{2} * \log n$. Головним *недоліком* є те, що не всі можливі комбінації зв'язків між процесорами можуть бути реалізовані. Тут можуть виникати блокування, для запобігання яким треба використовувати спеціальні концепції програмування. Блокування суттєво затримує виконання паралельної програми, що зумовлено необхідністю переконфігурації комутуючої сітки та побудови нової схеми сполучень між процесорами. Ця ситуація аналогічна тій, що має місце в телефонних мережах.

Комутуючі мережі Клоса

Об’єднання кращих властивостей розподілювача перехресних шин та дельта-сіток зроблено в комутуючих мережах Клоса. Вимогою до цієї мережної структури є забезпечення довільної комбінації зв'язків між процесорами, тобто не допускається поява блокувань. З другого боку, витрати на реалізацію (кількість пунктів комутації) мають бути мінімальними. Це досягається побудовою мінімізованої за витратами багатоступеневої мережі. Причому елементи одного ступеню будуються на основі простіших малих розподілювачів перехресних шин. На рис.2.6 показано триступеневу мережу Клоса з N=12 входами, розподіленими на a=4 групи, кожна з яких має m=3 входів.

У триступеневій мережі Клоса загальна кількість *N* ліній, що перемикаються повністю, реалізується в першій ступені *a* малими розподілювачами перехресних шин, кожний з яких має

m входів і *2*(m-1)* виходів *(N=a*m)*. Друга ступінь побудована на *2*(m-1)* розподілювачах перехресних шин, кожний з яких має *a* входів і *a* виходів, а третя - на розподілювачах з *2*(m-1)* входами і *m* виходами. Параметром, який може вибиратися під час конфігурування мережі Клоса, є, таким чином, ***m*.** Загальна кількість пунктів комутації триступеневої мережі Клоса буде приблизно мінімальною, якщо *m* визначити за формулою:

$$m \approx \sqrt{N/2}$$

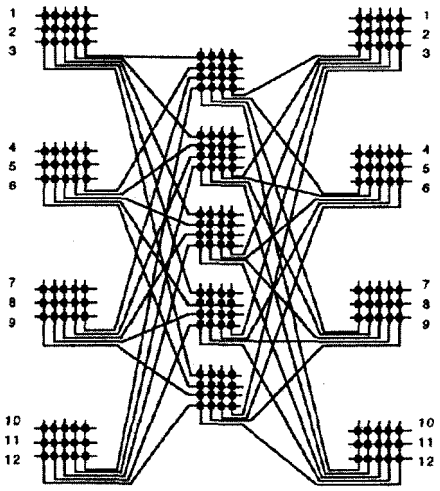


Рис. 2.6. Трикаскадна мережа Клоса для N=12 (при a=4, m=3)

При N≥ 24, мережа Клоса ефективніша, ніж простий розподілювач перехресних шин.

Загальна кількість пунктів комутації в триступеневій мережі Клоса визначається за формулою:

$$K = m*(2*m-1)*a + a^2*(2*m-1) + (2*m-1)*m*a = m*(2*m-1)*2*a + a^2*(2*m-1)$$

Вона містить (2*a) розподілювачі перехресних шин розміром *"m:(2*m-1)"* на ступенях 1, 3 і *(2*m-1)* розподілювача розміром "*a:a*" на ступені 2. З урахуванням того, що *N=a*m,* маємо:

$$\Rightarrow K = (2*m-1)*\left(\frac{N^2}{m^2} + 2*N\right)$$

Якщо кількість входів *m* оптимальна, то витрати на триступеневу мережу Клоса становлять приблизно $K \approx \sqrt{32}*N^{3/2}$.

На противагу цьому витрати на повний розподілювач перехресних шин відповідного розміру становлять приблизно N^2.

Приклад.

Для триступеневої мережі Клоса з 1000 входами та виходами справедливо:

$m \approx \sqrt{1000/2} \approx 22.4$

Беремо найближче *m=20*, що дає *a=N/m*=50.

Загальна кількість пунктів комутації буде:

$$K = 39*\left(\frac{1000000}{400} + 2000\right) = 175500$$

Ця кількість значно менша, ніж у повному розподілювачі перехресних шин цього розміру:

$K_{KS} = N^2 - N = 999000$

Таким чином, мережа Клоса заощаджує понад 82% пунктів комутації порівняно з розподілювачем перехресних шин. Розміри розподілювачів перехресних шин, що застосовуються як елементи мережі Клоса, вибирають відповідно до кількості комутованих процесорів і це виключає можливість блокувань зв'язків.

Мережі типу Fat-Tree (батьківське дерево)

Fat-Tree - решітчаста структура, що може масштабуватись як за кількістю процесорів, так і за кількістю можливих з'єднань, що відбуваються одночасно. Процесори - це листя повного двійкового дерева, в той час як внутрішні вузли - це ключові елементи (рис.2.7).

Кількість процесорів - $N=2^m$, а кількість ключів перемикання дорівнює *N-1*. Чим вище міститься ключовий елемент у дереві, тим більше ліній зв'язку перемикається в ньому (звичайно, немає потреби у подвоєній кількості ліній на кожному рівні дерева).

Комутаційна структура Fat-Tree близька до оптимальної. Ця властивість робить її дуже цікавою для застосування в паралельних структурах.

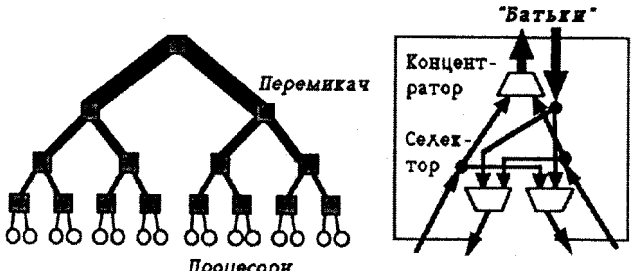


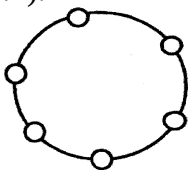
Рис. 2.7. Мережа типу Fat-Tree та перемикач

4. Структури, що забезпечують зв'язок типу "пункт-пункт"

Розгляньмо статичні структури зв’язку "пункт-пункт".

Кільце

Кільцева структура (рис.2.8) має дві лінії зв'язку на кожний ПЕ (*перевага*) і потребує в найгіршому випадку *n/2* кроки для обміну даними між двома ПЕ, що розміщуються в кільці на найбільшій відстані один від одного (*недолік*).

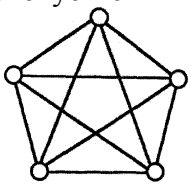


V=2
A=n/2

Рис. 2.8. Кільце

Повний граф

Повний граф (рис.2.9) має оптимальну коннективність (кожний ПЕ зв’язаний безпосередньо з будь-яким іншим ПЕ) забезпечується *n-1* лінією зв'язку на кожний ПЕ.

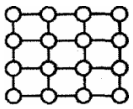


V=n-1
A=1

Рис. 2.9. Повний граф

Решітки і тори

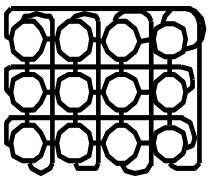
Дуже часто застосовуються решітчасті структури зв'язку і їхні замкнуті варіанти - тори. На рис.2.10 показано різницю між чотири - і восьми - зв'язковими структурами.



квадратна решітка
(4 лінії зв’язку в ПЕ)

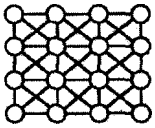
V=4; A=2* $\sqrt{n}$ -2 = 2* $\sqrt{n}$

Кваратний тор (4 лінії зв'язкуПЕ)



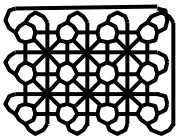
V=4; A= $\sqrt{n}$

V=8; A= $\sqrt{n}$ -1= $\sqrt{n}$



квадратна решітка

Кваратний тор (8 лінії зв'язку ПЕ)



V=8 A= $\sqrt{\frac{n}{2}}$

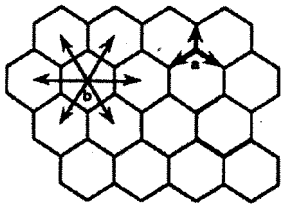
(8 ліній зв’язку в ПЕ)

Рис. 2.10. Квадратні решітки та тори

Усі квадратні решітки мають максимальну відстань між ПЕ, що дорівнює квадратному кореню від кількості ПЕ. Збільшення кількості ліній вдвічі у восьмизв'язковій решітці робить відстань між ПЕ наполовину меншою. Такий самий ефект досягається переходом до замкнутого тора, в якому кількість ПЕ залишається без зміни.

Гексагональна решітка.

Гексагональна решітка може розглядатись як видозміна квадратної решітки. Залежно від того, де розміщені процесорні елементи (на перехресті чи в центрі, рис.2.11 а, б), вони потребують 3 або 6 ліній зв'язку.



а) ПЕ в кутку

V=3

$A=2*\sqrt{n}$

б) ПЕ в центрі

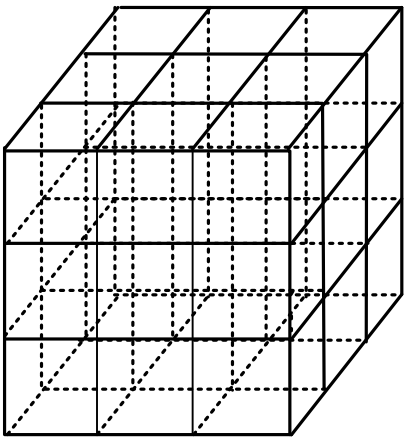
V=6

$A=3/2*\sqrt{n}$

Рис. 2.11. Гексагональна решітка

Кубічна решітка

У кубічній решітці (рис.2.12) зроблено перехід від двовимірної до тривимірної решітки. Процесорні елементи розміщені в просторі куба потребують 6 ліній зв'язку з ПЕ - сусідами. Максимальна відстань між ПЕ тут зменшується, вона пропорційна $\sqrt[3]{n}$.

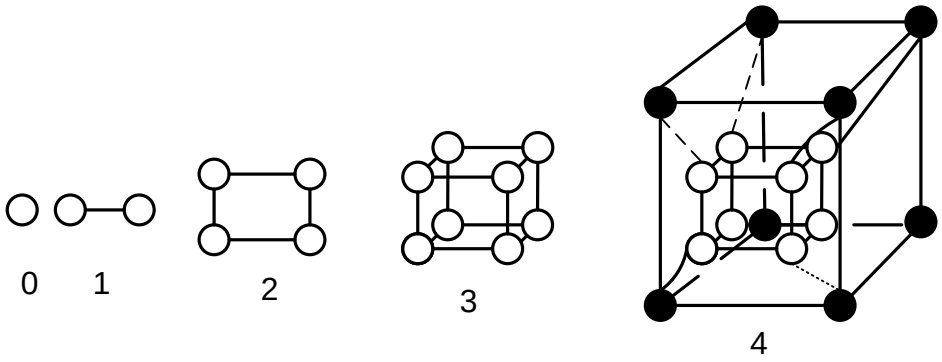


$V=6;\ A=3*\sqrt[3]{n}-3$

Рис. 2.12. Кубічна решітка

Гіперкуб

Гіперкуб нульової вимірності - це єдиний елемент (рис.2.13, ліворуч). Гіперкуб вимірності $i+i$ виникає з двох гіперкубів вимірності i**,** в яких елементи, що взаємодіють, з’єднані між собою. Наприклад, на рис.2.13 показано, що з двох квадратів (гіперкубів вимірності 2) можна побудувати куб (гіперкуб вимірності 3), з’єднавши кожний елемент "переднього" з кожним елементом "заднього" квадрата. Гіперкуб є універсальною мережею зв'язку з невеликою логарифмічною відстанню і не дуже великою логарифмічною кількістю ліній зв'язку у кожного ПЕ.



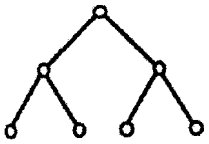
$n=2^m$; $V=log_2 n$

$A=log_2 n$

Рис.2.13. Гіперкуб з розмірностями від нуля до чотирьох

Двійкове дерево

Наступна структура зв'язку, що має логарифмічну відстань між ПЕ, це двійкове дерево (рис.2.14). Недоліком структури є "вузьке місце кореня", яке обмежує обмін даними між парами ПЕ з різних піддерев подібно до того, як це робить шинна система. Деякою мірою цей недолік зменшує застосування Fat-Tree.

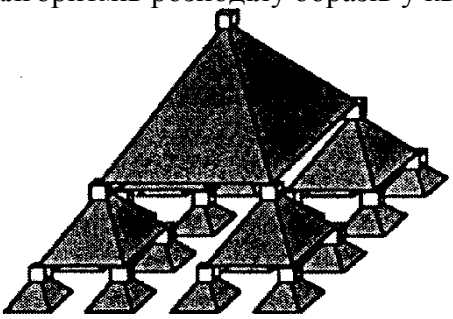


$N=2^m-1$
$V=3$
$A=2*\log_2 (n+1)-2 \approx 2*\log_2 n$

Рис. 2.14. Двійкове дерево

Пірамідальне дерево

У цій структурі зв’язку (рис.2.15) кожна вершина має 4 послідовники в дереві що, може використовуватися для побудови алгоритмів розподілу образів у квадрантах.



$N=(1/3)*(4^m -1)$
$V=5$
$A=2*\log_4 (3*n+1)-2 \approx 2*\log_4 n$

Рис. 2.15. *Пірамідальне дерево*

Ротація-зміна. (Shuffle-Exchange)

До структур з логарифмічною вимірністю належить також мережа Shuffle-Exchange (ротація, або циклічний зсув та заміна). Вона складається з двох розділених видів зв'язку - одностороннього "Shuffle" та двостороннього "Exchange" (рис.2.16).

$N=2^m$

$V=2$ (1 двонаправлена і 2 однонаправлені лінії)

$A=2log_2 n$

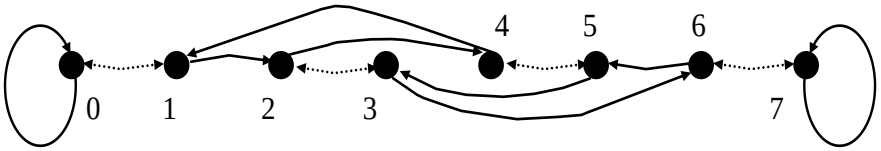


Рис.5.16. Мережа типу “Shuffle-Exchange”

Обидві структури зв'язку можуть бути дуже просто представлені за допомогою операції відображення номерів ПЕ бінарним способом запису (p_i,-біт бінарного номера ПЕ, $i=i,2,..m$):

Shuffle ($p_m, p_{m-1}, ..., p_1$)=(p_{m-1}, , p_1, p_m) - одна ротація вліво бінарного номера ПЕ (*m* разів);

Exchange (p_m, p_1)=(p_m, p_2 p_1) - заміна молодшого біта бінарного номера ПЕ операцією заперечення.

Shuffle - частина з'єднує кожний ПЕ з тим ПЕ, бінарний номер якого обчислюється циклічною операцією ротації вліво. Перший та останній елементи, що мають однакові двійкові розряди в номерах (на рис.2.16 це ПЕ *No→ 000, ПЕ* N_7 *→111*) відображаються самі на себе, тоді як інші ПЕ з'єднані в цикли.

Exchange - зв'язок у двійковій формі запису заперечує наймолодший біт ПЕ, тобто Exchange з'єднує двонаправлено кожний парний ПЕ з своїм правим сусідом.

Приклад застосування операцій Shuffle та Exchange:

1) *001→010→100→001*

2) *011→010→011* двосторонній зв'язок ПЕ3 ↔ ПЕ2

Плюс-мінус 2^i*- мережа (PM2i)*

Мережа дещо складніша. При наявності $n = 2^m$ вузлів (вершин) мережі (процесорних елементів ПЕ) вона складається з *2*m-1* окремих структур зв'язку, які позначаються як:

$$PM_{+0}, PM_{-0}, PM_{+1}, PM_{-1}, PM_{+2}, PM_{-2},..., PM_{m-1},.$$

Для *PMs* справедливі такі визначення:

$$PM_{+1}(j)=(j+2^I)\ mod\ n;$$
$$PM_{-1}(j)=(j-2^I)\ mod\ n.$$

Індекс кожної односторонньої структури показує, якою має бути відстань до сусідньої вершини (у двійковому порядку). Для PM_{+0} відстань буде +2° =+1, для PM_{-0} відповідно -2° =-1 Відстань між вершинами для PM_{+2} при цьому є $+2^2$ = +4, для PM_{-2} буде - 4. Для вищого показника *m-1* з урахуванням замкнутості структури має місце співвідношення: $PM_{+(m-1)}=PM_{-(m-1)}$

Обидві структури ідентичні, тому є тільки *2*m-1*, а не *2*m* структур.

На рис.2.17 показано мережу *PM2I* з n=8 ПЕ. PM_{+0} і PM_{-0} зв'язують кожний ПЕ з своїм правим або лівим сусідом і створюють таким чином двостороннє кільце зв'язку. , PM_{+1}, і PM_{-1}, включають дві розділені односторонні структури по чотири ПЕ. Кожний ПЕ має сусіда з номером через один, причому *Modulo* - операція утворює структуру зв'язку типу "кільце". Разом вони утворюють два окремих двосторонніх кільця. Остання структура PM_{+2} ідентична з PM_{-2},. Кожний ПЕ зв'язаний тут з іншим ПЕ на відстані 4 (*Modulo 8*).

$V=2*\log_2 n-1$ $A=\log_2 n-1$

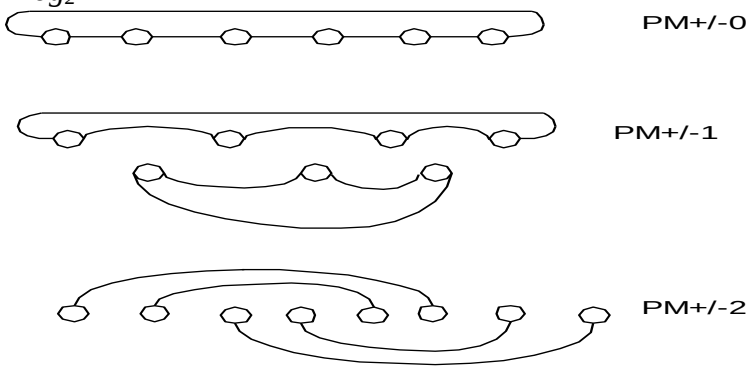


Рис. 2.17. Мережа типу PM2I

Порівняння мереж

Порівняння мереж можливе тільки на основі параметрів *V* і *A*. Переваги, що має певна мережа, можна отримати тільки з урахуванням особливостей паралельної задачі. Якщо алгоритм розв'язування задач потребує певної мережі, то, як правило, фізична реалізація якраз цієї, можливо "гіршої" за параметрами *V* і *A* мережі, дасть кращі результати, ніж пристосування для цієї задачі іншої мережі, що має "хороші" параметри.

Для вирішення кожної проблеми неможливо будувати спеціалізовану систему. Кожна паралельна система має у своєму розпорядженні декілька комутаційних структур, які можуть бути багатосторонніми і здатними до динамічної конфігурації. Усі алгоритми, які підлягають імплементації в цій паралельній системі, мають задовольнятися наявною мережею (або її частиною). Цінність мережі залежить, таким чином, від того, наскільки якісно вона може використовуватись в середньому для вирішення проблем, що постають найчастіше. Наприклад, паралельна ЕОМ MasPar MP-1 функціонує із застосуванням двох різних мереж: решітчастої структури та триступеневої мережі Клоса, що забезпечує хорошу загальну коннективність. Алгоритми, які потребують решітчастої структури, можуть використовувати безпосередньо цю швидку локальну структуру, тоді як алгоритми, яким потрібні інші мережні структури, можуть бути реалізовані з використанням маршрутизації на глобальній мережі Клоса з деякими втратами ефективності.

Зігель моделював одні мережі з застосуванням інших мереж. При цьому виявляється передусім перевага Shuffle-Exchange та гіперкуба *як* "універсальних мереж", тобто як засобів ефективної емуляції декількох різноманітних мережних структур. Решітка тільки тоді виправдана, коли решітчаста структура потрібна для реалізації алгоритмів; для моделювання інших мереж вона непридатна. Насамперед емуляція мережі може бути виконана автоматично компілятором лише тоді, коли крім мережної структури цільової системи відома також структура, яка потрібна для реалізації прикладної програми.

Найчастіше реляції про міжпроцесорні зв'язки, що мають забезпечити обмін даними, задані у вигляді складних арифметичних виразів, мало відрізняються від "стандарту" або взагалі обчислюються і стають відомими лише перед запуском програм. В табл.2.1 показано кількість потрібних циклів моделювання залежно від кількості її процесорних елементів.

Таблиця 2.1. Порівняння комутаційних мереж

Мережа Моделює Мережу	Двовимірна решітка	PM2 I	Shuffle-Exchange	Гіперкуб
Двовимірна решітка	—	$\approx \sqrt{n}/2$	$\approx \sqrt{n}$	$\sqrt{n}$
PM2I	1	—	$\approx \log_2 n$	2
Shuffle-Exchange	$\approx 2\log_2 n$	$\approx 2\log_2 n$	—	$\approx \log_2 n+1$
Гіперкуб	$\log_2 n$	$\log_2 n$	$\log_2 n$	—

5. Методи комутацій

Важливим в організації роботи комутаційної мережі є метод перемикання, який визначає як повідомлення передаються від вузла-передавача до вузла-приймача. Передача здійснюється *повідомленнями* і *пакетами*.

Виділяють три основних методи комутації в комутаційних мережах паралельних систем:

- комутація з проміжковим зберіганням (“зберігання- і передача” – Store-and-forward);
- комутація каналів;
- комутація віртуальних каналів.

В першому випадку повідомлення повністю приймається на проміжному вузлі і тільки після цього передається далі. Пересилка виконується як тільки звільнився черговий канал передачі, тоді повідомлення передається на наступний вузол. Буферизація вимагає додаткової пам’яті і затрат часу. Метод використовується тоді, коли час відклику не суттєвий.

При комутації каналів між джерелом і одержувачем повідомлення встановлюється неперервний зв'язок, після чого виконується передача даних. Комутуючий канал встановлюється тільки на час з’єднання . Приклад – телефонний зв'язок.

Зменшити затримки при передачі даних дозволяє метод віртуальних каналів. В цьому випадку пакети накопичуються в проміжних вузлах тільки тоді, коли недоступний черговий канал зв’язку. В іншому випадку пересилка виконується негайно і без буферизації.

6. Загальні зауваження стосовно оцінки продуктивності паралельних систем

Паралельні системи характеризуються: *різноплановістю задач, типом опрацювання (векторне, скалярне), різними операційними системами, різними конфігураціями систем, одночасністю виконання операцій, складністю взаємодії між елементами системи,* що не дозволяє використовувати стандартні підходи до визначення їх продуктивності.

А продуктивність окремого процесора залежить від: *частоти синхронізації, середньої кількості тактів на команду, кількості команд, що виконується.*

Тому *час виконання деякої програми в процесорі може бути виражений двома способами: кількістю тактів синхронізації для даної програми, які перемножуються на тривалість такту синхронізації, або кількістю тактів синхронізації для даної програми поділеними на частоту синхронізації.*

В процесі пошуку стандартної одиниці вимірювання продуктивності комп'ютерів було прийнято декілька популярних одиниць вимірювання, а для оцінки продуктивності паралельних вузлів деякі терміни були штучно вирвані з їх контексту і використані там, для чого вони ніколи не призначалися. Насправді *єдиною і надійною одиницею вимірювання продуктивності є час виконання реальних програм*, і всі пропоновані заміни цього часу як одиниці вимірювання або заміни реальних програм як об'єктів вимірювання на синтетичні програми тільки вводять в оману.

Небезпеки використання популярних альтернативних одиниць вимірювання (MIPS і MFLOPS) для оцінки продуктивності паралельних систем.

MIPS - *швидкість виконання операцій за одиницю часу,* тобто - відношення кількості команд в програмі до часу її виконання. *MIPS:*

- *залежить від набору команд процесора, що затруднює порівняння за показниками MIPS комп'ютерів, що мають різні системи команд;*
- *навіть на тому ж комп'ютері змінюється від програми до програми;*
- *змінюватися по відношенню до продуктивності в протилежний бік (при більшому значенні MIPS реальна продуктивність менша).*

Приклад для останнього випадку.

До складу процесора входять *вузли обчислення елементарних функцій*. За відсутності *вузлів* операції обчислення елементарних функцій виконуються за допомогою підпрограм. Тоді, такі процесори мають вищий рейтинг MIPS, але виконують більшу кількість команд, що приводить до збільшення часу виконання програми.

MFLOPS - мільйон операцій з рухомою крапкою за секунду. *Як одиниця вимірювання, MFLOPS, призначена для оцінки продуктивності тільки операцій з рухомою крапкою, і тому не застосовується поза цією обмеженою областю.*

Наприклад, програми компіляторів мають рейтинг MFLOPS близький до нуля не залежно від того, наскільки швидка машина, оскільки компілятори рідко використовують арифметику з рухомою крапкою.

Рейтинг MFLOPS залежить і від характеристик процесора і від програми, і є об'єктивнішим ніж параметр MIPS, оскільки *базується на кількості виконаних операцій, а не на кількості виконаних команд.*

Проте і з використанням MFLOPS для оцінки продуктивності виникають певні проблеми. Перш за все, це пов'язане з тим, що *набори операцій з рухомою крапкою не сумісні на різних комп'ютерах.* Наприклад, в суперкомп'ютерах фірми Cray Research відсутня команда ділення (є, правда, операція обчислення оберненої величини числа з рухомою крапкою, а операція ділення може бути реалізована за допомогою множення діленого на зворотну величину дільника). В той же час сучасні мікропроцесори мають команди ділення, обчислення квадратного кореня, синуса і косинуса, тощо.

Інша проблема полягає в тому, що *рейтинг MFLOPS міняється не тільки на суміші цілочисельних операцій і операцій з рухомою крапкою, але і на суміші швидких і повільних операцій з рухомою крапкою.* Наприклад, програма з 100% операцій додавання матиме вищий рейтинг, ніж програма з 100% операцій ділення.

Для усунення цих недоліків *використовується "нормалізоване" число операцій з рухомою крапкою*, за тестовим пакетом "Ліверморські цикли" (див. табл.2.2.).

Таблиця 2.2. Співвідношення між реальними і нормалізованими операціями з рухомою крапкою

Реальні операції з рухомою крапкою	Нормалізовані операції з рухомою крапкою
“+” “–” “x” “порівняння”	1
“ділення” “√”	4
“exp” “sin”	8

Поява векторних і паралельних процесорів і систем, не зменшила важливості Ліверморських циклів, проте змінилися значення продуктивності і величини розкиду між різними циклами. На векторних структурах продуктивність залежить не тільки від елементної бази, але і від характеру самого алгоритму, тобто коефіцієнта векторизації.

На паралельній машині продуктивність істотно залежить від відповідності між структурою апаратних засобів і структурою обчислень в алгоритмі. Важливо, щоб тестовий пакет представляв алгоритми різних структур. Тому в Ліверморських циклах зустрічаються послідовні, сіткові, конвеєрні, хвильові обчислювальні алгоритми, що підтверджує їх придатність і для паралельних машин.

7. Фактори, що необхідно враховувати при оцінці продуктивності

При оцінці продуктивності необхідно враховувати: *тип алгоритму, тип програмного забезпечення, параметри протоколів каналів передачі даних, структуру окремого процесора.*

Тип алгоритму. Алгоритм, що ідеально пристосований для роботи на одній архітектурі, на іншій (з цією ж кількістю процесорів) може працювати набагато гірше. Для масивно-паралельних систем необхідний масштабований алгоритм (для “оптимального” завантаження всіх процесорів). Виграш дає оптимальне поєднання “структура - алгоритм”. Тобто, *на паралельній машині продуктивність залежить від відповідності між структурою апаратних елементів і структурою обчислень в алгоритмі. Не можна переносити результат з однієї на іншу систему.*

Тип програмного забезпечення. Програмне забезпечення (ПЗ) паралельних структур має певні особливості, а саме: *вартість програм є високою, при перенесенні програми з однієї машини на іншу необхідна доробка програми, всі системи відлагодження програми впливають на її поведінку (наприклад, покрокове відлагодження для паралельних систем неефективне).* Крім того, *програмісту важко навчитися мислити паралельними категоріями.*

До складу ПЗ необхідно включати *процедури маршрутизації.* Для оцінки продуктивності розподіленої системи необхідно знати: *топологію зв’язків, швидкість виконання арифметичних операцій, час ініціалізації каналу зв’язку, час передачі одиниці інформації.*

Крім того, потрібно пам’ятати, *що ріст продуктивності процесорів випереджає ріст швидкості комутаційних каналів.*

Протокол каналів передачі. Для паралельних машин доцільно визначити бібліотеку передачі повідомлень, яка враховує особливості машини. В 1994 році прийнятий стандартний інтерфейс передачі повідомлень *MPI (Message Passing Interface Standart)* – процедурний інтерфейс для мов C і Fortran). Інтерфейс визначає всі функції, необхідні для передачі повідомлень “точка-точка”. Для колективних повідомлень вводяться поняття *групи процесорів* (з якими можна оперувати як з кінцевими множинами), *комунікатора* (реалізують контекст для передачі повідомлень). Забезпечує трансляцію повідомлень з форми одного процесора у вид, який необхідний іншому процесорові. Не вирішена проблема динамічного балансування.

8. Методи оцінки продуктивності паралельних систем

Є такі методи оцінки продуктивності паралельних систем: *метод обчислення продуктивності складових частин, метод експертних оцінок, розрахунковий метод, практичний метод.*

Метод обчислення продуктивності складових частин. Для паралельних систем неефективний.

Метод експертних оцінок. Найскладніший і найзаперечливіший з усіх методів. Розроблений консорціумом стандартизації методів всесторонніх оцінок системи (PC Bench Concortium). Особлива увага приділена *апаратному підходу* (стараються наслідувати ідеалізовану тестову модель – виключити вплив сторонніх систем, які в даний момент не тестуються).

Використовується *дві групи тестів:*

- *синтетичні* (вимірюють швидкість роботи і базуються на хронометражі роботи реальних застосувань; легко ізолюють процесор від решти компонентів системи. До реальної задачі можна віднести з натяжкою);
- *виключно процесорні тести* (для процесорів окремо).

Розрахунковий метод. Базується на обчисленні продуктивності. Є трудомістким і недосконалим.

Практичний метод. Розв'язання конкретної задачі на конкретній структурі. Дає об'єктивну оцінку продуктивності. *Недолік* – продуктивність можна оцінити тільки після виготовлення системи. При негативному результаті – висока ціна помилки.

9. Характеристики продуктивності паралельних алгоритмів.

Характеристиками продуктивності паралельних алгоритмів є: *фактор прискорення, максимальне прискорення (закон Амдала), ефективність паралельного алгоритму, ціна, масштабність, загальний час виконання паралельного алгоритму, повний час виконання паралельного алгоритму, теоретичний час комунікацій.*

Оцінюючи продуктивність безпосередньо на паралельних системах, відзначають позитивний ефект від *розпаралелювання (Speedup*- прискорення) і вигоди від збільшення *масштабності розв'язаних задач (Scaleup)*.

Показник Speedup визначає, у скільки разів швидше може бути вирішена одна й та ж задача на N процесорах порівняно з її вирішенням на одному процесорі.

Показник Scaleup визначає, у скільки разів більшу за розмірами проблему можна вирішити за той же час N процесорами порівняно з проблемою, що вирішується одним процесором.

Амдалом сформульований "закон Амдала", де єдиним параметром є поділ програми на послідовну і паралельну частини; масштаб вирішуваної задачі залишається постійним. Розгляньмо дещо спрощену формулу цього закону.

Фактор прискорення. Прискоренням (*speedup factor*) паралельного алгоритму в N – процесорній системі називається величина $S(N) = T_1/T_N$, де

- T_1 – час виконання алгоритму на одному процесорі чи однопроцесорній системі;
- T_N – час виконання алгоритму на багатопроцесорній системі з N - процесорами.

Закон Амдала. Позначимо:

- P_c- максимальний ступінь розпаралелення (максимальна кількість процесорів, які можуть працювати паралельно в будь-який момент часу в період виконання програми). Тут вирішальне значення має також тип застосовуваної моделі паралельності (наприклад, MIMD чи SIMD);
- T_k- тривалість виконання програм з максимальним показником розпаралелення $P_c \geq k$ на системі з k процесорами;
- N - кількість процесорів у паралельній системі;
- f – відсоток послідовних операцій програми (не можуть бути виконані паралельно на N процесорах).

Тривалість виконання програми на паралельній системі з N процесорами оцінюється формулою:

$$T_N = f * T_1 + (1 - f) * \frac{T_1}{N},$$

звідки дістанемо показник прискорення (Speedup) в системі з *N* процесорами

$$S_N = \frac{T_1}{T_N} = \frac{N}{1 + f * (1 - N)}.$$

Оскільки, *0< f < 1*, справедливе таке співвідношення:

$$1 \le S_N \le N,$$

тобто показник прискорення не може бути більшим ніж кількість процесорів *N*.

Як міра досягнутого прискорення, відносно максимального визначається ефективність системи з N процесорами

$$E_N = S_N / N$$

Область межі ефективності : $1/N \le E_N \le 1$

На практиці використовують значення E_N у відсотках. При E_N = 0,9. наприклад, могла б бути досягнута ефективність, що дорівнює 90% максимально можливої.

Приклади застосування закону Амдала

1. Система має N=1000 процесорів

• Програма має максимальний показник розпаралелення 1000

• 0,1% програми виконується послідовно (наприклад, операції вводу-виводу), тобто f=0,001 Обчислення показника прискорення дають:

$$S_{1000} = \frac{1000}{1 + \frac{1000 - 1}{1000}} \approx 500$$

Таким чином, незважаючи на суттєво малу долю послідовної частину програми, у цьому випадку досягається тільки половина максимально можливого прискорення, тобто $S_{\max} = 1000$. Ефективність буде $E_{1000} = 50\%$.

2. Система має N=1000 процесорів

• Програма має максимальний показник розпаралелення 1000.

• 1% програми має виконуватися послідовно, тобто f=0,01

Показник прискорення:

$$S_{1000} = \frac{1000}{1 + \frac{1000 - 1}{100}} \approx 91$$

Ефективність E_{1000} = 9,1%. тобто використовується тільки 9,1% загальної потужності процесорів і це при малій, на перший погляд, послідовній частині програми!

З збільшенням послідовної частини програми падає завантаження процесорів, а з ним і показник прискорення порівняно з послідовною обчислювальною системою. Для кожної послідовної частини програми може бути обчислений максимально можливий показник прискорення незалежно від кількості застосовуваних процесорів:

$$\lim_{N \to \infty} S_N(f) = \frac{N}{1 + f * (N - 1)} = \frac{1}{f}$$

Це означає, що програма із скалярною частиною, яка становить 1%, ніколи не зможе досягти показника прискорення (Speedup), що був би більшим 100 - незалежно від того, застосовується 100,1000 або 1000000 процесорів.

На рис.2.18 наведені графіки залежності показника прискорення від кількості процесорів *N* для різних величин послідовної частини програми f. Головна діагональ належить до лінійного прискорення і може бути побудована при f=0.

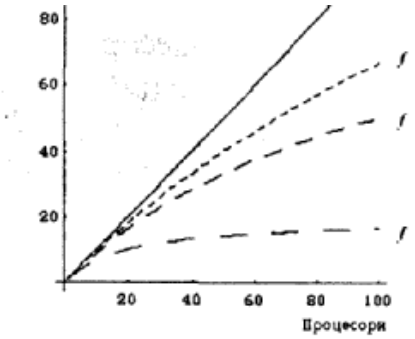


Рис. 2.18. Залежність прискорення від кількості процесорів

На рис. 2.19 наведені графіки залежності ефективності від *N* і f.

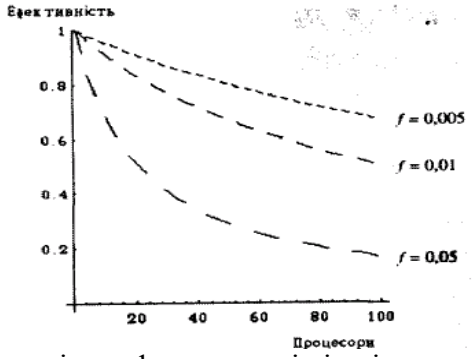


Рис.2.19. Залежність ефективності від кількості процесорів.

Ефективність паралельного алгоритму *E* показує у скільки разів більший час виконання завдання одним процесором T_1, ніж час виконання цього ж завдання багатопроцесорною системою T_N, помножений на кількість процесорів *N*. $E = T_1/(T_N*N) = S_N/ N$. Ефективність характеризує ту частину часу, яку процесори використовують на обчислення.

Ціна (робота) обчислення визначається як *Cost = (час виконання) * (повне число використаних процесорів).* Ціна послідовних обчислень рівна часу їх виконання T_1. Ціна паралельних обчислень рівна T_N*N**.** З іншого боку $T_N = T_1/S_N$. Звідси ціна паралельних обчислень: $Cost = T_1*N/ S_N = T_1/ E$ **.** Ціноптимальні паралельні алгоритми – алгоритми, в яких ціна для вирішення проблеми на багатопроцесорній системі є пропорційною ціні (часу виконання) на однопроцесорній системі.

Масштабність (Scaleup)

Оскільки для вимірювання ефективності застосовується та сама програма, то показник *f* в законі Амдала залишається незмінним. Проте, кожна паралельна програма, незалежно від її величини, має послідовно обробляти деяку постійну мінімальну кількість *f* своїх інструкцій. Це означає, що графіки, які наведені на рис.2.18 і рис. 2.19 із зміною розмірів проблеми стають недійсними! Практичні заміри на паралельних системах з дуже великою кількістю процесорів показали: чим більшими є вирішувана проблема і кількість використовуваних процесорів, тим меншою стає відсоток послідовної частини обчислень. Це дає підставу зробити висновок, що на паралельній системі з дуже великою кількістю процесорів можна досягти високої ефективності.

Повний час виконання паралельного алгоритму t_p є сумою часу, що витрачається на обчислення t_{comp}, і часу, що витрачається на комунікації (обмін даними між процесорами) t_{comm}. Час обчислень оцінюється як час для послідовного алгоритму. $t_p=t_{comp}+t_{comm.}$ Позначимо час запуску (startup), що інколи називають часом скритого стану повідомлень (message latency), як $t_{startup}$. Як початкову апроксимацію *часу комунікації* візьмемо: $t_{comm}=t_{startup}+n * t_{data}$. Будемо рахувати $t_{startup}$ постійним і залежним як від обладнання, так і від програмного забезпечення. Час передачі одного слова даних t_{data} також вважатимемо постійним. Нехай від одного до другого процесора передаються *n* даних, *тоді теоретичний час комунікацій* можна представити в виді графіка (див.рис.2.20).

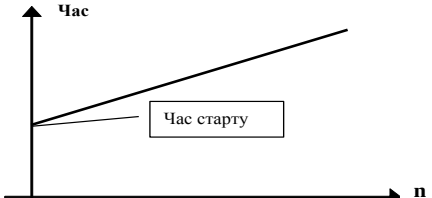


Рис.2.3. Теоретичний час комунікацій

Для передачі q повідомлень, кожне з яких має довжину n-даних, необхідний час $t_{comm}=q(t_{startup}+n\ t_{data})$**.** Інформативною величиною є також відношення обчислювальних затрат до комунікаційних: t_{comp}/t_{comm}.

Визначення для варіантів програми, що мають різну величину

$T_k(m)$ - тривалість виконання програми з величиною проблеми m і максимальним показником розпаралелення $P_c \geq k$ на k процесорах.

Показник масштабованості деякої проблеми, величина якої n на k процесорах порівняно з меншою проблемою m (m<n) на одному процесорі визначається так:

якщо $T_1(m) = T_k(n)$, тобто тривалість виконання "малої" програми на одному процесорі дорівнює тривалості виконання "великої" програми на k процесорах, то показник масштабності можна виразити формулою:

$$SC_k = \frac{n}{m}$$

Зауважимо, що тривалість виконання залежить від такого параметра, як "величина проблеми", яка точно не визначена. Її в даному аспекті можна трактувати як кількість даних, які обробляються варіантами програм різної величини за одним і тим же алгоритмом.

На практиці із збільшенням кількості процесорів найчастіше вирішуються більші проблеми і в більшості практичних застосувань не ставиться задача вирішувати ті ж проблеми швидше за рахунок збільшення кількості процесорів. У таких практичних областях показник *Scaleup* має більше значення, ніж *Speedup*.

Графік залежності показника збільшення складності вирішуваних задач SCк від кількості процесорів наведений на рис.2.21.

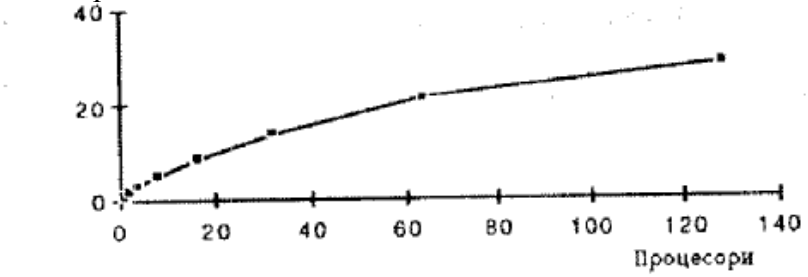


Рис.2.21 Залежність показника збільшення складності вирішуваних задач SCк від кількості процесорів

10. Порівняння MIMD і SIMD структур за продуктивністю

Вище, при аналізі продуктивності паралельних обчислювальних систем не розрізнялись MIMD- та SIMD- системи. Проте кожна програма містить у собі як мінімум два різних максимальних показника паралельності: один для асинхронної паралельної обробки, P_{MIMD} , а другий - для синхронної, P_{SIMD}. У зв'язку з універсальністю асинхронної паралельної обробки має місце співвідношення $P_{SIMD} \leq P_{MIMD}$.

На практиці необхідно звернути увагу на такі моменти:

- якщо в MIMD-системі є вільні процесори, що не використовуються в даній задачі, то вони можуть бути застосовані іншими користувачами для своїх задач, що розв'язуються одночасно. В SIMD-системах неактивізовані процесори не використовуються;

- спосіб обробки інформації в SIMD-програмах дає суттєво менші показники розгалуження;

- процесорні елементи SIMD- систем, як правило, менш потужні, ніж процесори, що застосовуються в MIMD-системах, але цей недолік компенсується за рахунок значно більшої кількості процесорів SIMD- систем порівняно з MIMD-системами.

У MIMD-системах часто програмуються задачі в “SIMD-режимі”, тобто не використовується можлива незалежність окремих процесорів. Особливо це помічається, коли задача, що розв’язується, розподіляється між великою кількістю процесорів, тобто коли мова йде фактично про масивну паралельність. Ця обставина є одним з найвагоміших аргументів на користь SPMD- моделі паралельності (same program, multiply data), яка виникає внаслідок змішаного використання SIMD і MIMD в рамках однієї системи.

У зв'язку з тим що за законом Амдала ефективність сильно залежить від кількості процесорів, а з іншого боку, не має можливості уникнути деякої послідовної частини програми (як мінімум, залишаються програми вводу-виводу даних), постає питання: чи доцільно взагалі використовувати паралельну SIMD-систему з кількістю процесорів?

Якщо помилково застосувати закон Амдала, то можна одержати неправильні результати, до яких насамперед приводить спосіб обліку інструкцій (рис.2.5). У той час як в програмі А (наприклад, в MIMD- програмі) відповідно до формули Амдала враховується кожна елементарна операція (це дає фактор f_A), в програмі В для SIMD-системи як скалярні, так і векторні операції враховуються як одна інструкція (це дає фактор f_B). Таке визначення є природним для SIMD-системи, бо кожна операція, що так враховується, потребує приблизно однакового часу на виконання. Отже, маємо:

- f_A - послідовну частину програми відносно кількості елементарних операцій;

- f_B - послідовну частину програми відносно тривалості виконання операцій.

На рис.2.5 SIMD-програма може працювати паралельно половину свого часу (f_B =0,5). Водночас кількість послідовно виконуваних елементарних операцій суттєво менша, а саме f_A =1/6. Залежність між цими двома факторами виражається формулою

$$f_A = \frac{f_B}{N*(1-f_B)+f_B}.$$

Для отримання точних даних продуктивності, можна застосувати для кожної SIMD-інструкції виражену у відсотках кількість активних в цій інструкції процесорів відносно їхньої загальної кількості: це буде число між нулем (у випадку виконання скалярної операції в керуючій ЕОМ) та одиницею (випадок, коли всі ПЕ активні).

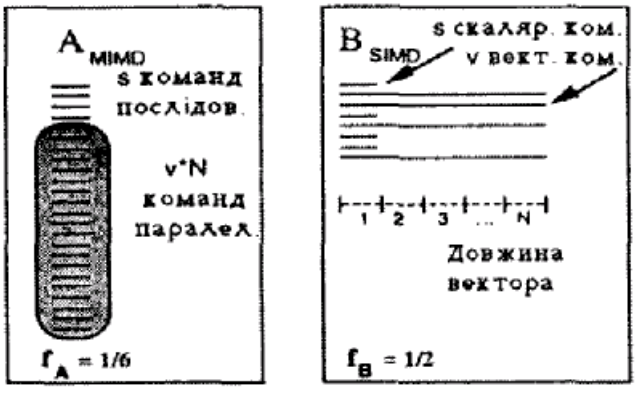


Рис. 2.22 Оцінка кількості паралельних інструкції

Для визначення показника прискорення SIMD-системи можна перерахувати T_N, на T_1, відповідно до запитання: "Скільки часу було б потрібно послідовній ЕОМ для виконання цієї паралельної програми?".

Визначення:

- f_B – частина скалярних інструкцій (у відсотках) SIMD-програми відносно загальної кількості (скалярних і векторних) інструкцій.

Можна вважати ідентичним і таке визначення

- f_B: частина часу виконання послідовних операцій (у відсотках) відносно загальної тривалості виконання паралельної програми.

Тоді,

$$T_1 = f_B * T_N + (1 - f_B) * N * T_N$$

$$S_N = \frac{T_1}{T_N} = f_B + (1 - f_B) * N$$

Приклади застосування зміненого закону

1.

- Система має 1000 процесорів.
- Програма має максимальний показник розпаралелення 1000.
- • 0,1 % програми (в SIMD-обчисленні) виконується послідовно, тобто f = 1/1000. Обчислення показника прискорення:

$$S_{1000}=0.001+(1-0.001)*1000\approx 999$$

У цьому випадку досягається прискорення, близьке до теоретично максимально можливого (E_{1000} =100%).

Приблизний результат одержуємо і за формулою Амдала з відповідним f_A .

$$f_A = \frac{f_B}{B*(1-f_B)+f_B} = \frac{0.001}{1000*(1-0.001)+0.001} = 10^{-6}\cdot$$

Показник прискорення за формулою Амдала:

$$S_{1000} = \frac{1000}{1+10*(1000-1)} = 999\cdot$$

2.

- Система має 1000 процесорів
- Програма має максимальний показник розпаралелювання 1000
- 10% програми (в SIMD-обчисленні) виконується послідовно, тобто f_B =0.1. Обчислення показника прискорення:

$$S_{1000} =0,1+0,9*1000=900.$$

Незважаючи на помітно велику послідовну частину SIMD-програми досягається 90% максимального прискорення!

Наведені тут міркування спрямовані на чітко окреслену проблему (з відповідно великим максимальним показником розпаралелення) і на задану незмінну кількість процесорних елементів: ні розміри задачі, ні кількість процесорів не змінюються, проводиться тільки порівняння з послідовною системою. У випадку, коли змінюється кількість ПЕ в процесі виконання програми, оцінити, як змінюється при цьому показник *speedup*, за наведеною формулою для S_N неможливо, бо вона базується на параметрі T_N ! Збільшення кількості ПЕ не привело б до швидшого виконання тієї самої SIMD-програми в області $N > P_{SIMD}$ (кількість ПЕ - більша або дорівнює максимальному показнику SIMD - розпаралелення), бо ці додаткові ПЕ були б зайвими і залишались би неактивними. Обчисливши всі T_i , де $1<i<N$, можна знайти відповідні показники прискорення S_i:

$$T_i = f_B T_N + (1 - f_B) * \frac{N * T_N}{i}; \qquad S_i = \frac{T_1}{T_i}$$

Наведена вище формула для S_N використана для екстраполяції тривалості виконання масштабованих варіантів проблеми з лінійним масштабним коеффіцієнтом (*Skaleup*) і визначена як "*scaled speedup*" (показник прискорення, пов'язаний з масштабами задач), що легко вводить в оману: тут досліджувались часові показники різних за масштабами варіантів програми і обчислені показники прискорення відносно цих варіантів. Було також прийнято, що послідовна частина (за формулою Амдала) деякої "реальної" програми при її масштабуванні залишається постійною, а збільшується лише паралельна частина. Проте за визначенням це є клас програм з лінійним показником масштабності (*scaleup*), тобто для нього справедлива формула

$$T_N(A) = T_k * n(K * A)$$

де k - число, яке показує, що в k разів більший варіант програми виконується в k разів більшою кількістю процесорів за той же час.

Крім того, завжди виконується умова:

$T_1(k * A) = k * T_1(A)$ - програма, більша в k разів, потребує при виконанні на одному процесорі в k разів більше часу.

З цього обмеженого класу задач, випливає лінійний приріст показника "*scaled speedup*":

$$\frac{S_{k*N}(k*A)}{S_N(A)} = \frac{\frac{T_1(k*A)}{T_{k*N}*k*A}}{\frac{T_1(A)}{T_N(A)}} = \frac{\frac{k*T_1(A)}{T_{k*N}(k*A)}}{\frac{T_1(A)}{T_N(A)}} = \frac{k*T_N(A)}{T_{k*N}(k*A)} = k.$$

Висновки

Усі без винятку дані, що характеризують продуктивність паралельних обчислювальних систем, мають розглядатися критично. Для цього є цілий ряд причин.

1. Характеристики продуктивності, як показник прискорення і показник масштабності завжди пов'язані з конкретним застосуванням паралельних систем - показники справедливі тільки для даного класу задач і дуже умовно можуть бути поширені на інші задач.

2. Показник прискорення деякої програми стосується тільки одного окремого процесора паралельної системи.

3. В SIMD- системах процесорні елементи, як правило, мають значно меншу потужність в порівнянні з MIMD-процесорами, що може дати на порядок, а то й більше, різницю в оцінках швидкості.

4. Завантаження паралельних процесорів - головна складність в MIMD-системах. У SIMD-системах це не так важливо, бо неактивні процесори не можуть бути використані іншими задачами. Незважаючи на це, показник прискорення обчислюється залежно від характеристик завантаження. Якщо SIMD-програма спробує "включити в роботу" непотрібні ПЕ, то дані завантаження, а з ними і показники прискорення, стануть недійсними. Паралельна програма в цьому випадку буде менш ефективною, ніж за результатами тестування.

5. Порівнюючи паралельну систему (наприклад, векторну) з послідовною (скалярною), не доцільно застосовувати два рази один і той же алгоритм (в паралельній і в послідовній версіях).

Приклад. OETS- алгоритм - типовий алгоритм сортування для SIMD-систем, але для послідовних процесорів найефективнішим є алгоритм Quicksort - в усякому разі Quicksort не може бути так просто переведений в ефективну паралельну програму для SIMD-систем. Порівняння "OETS- паралельного" і "OETS- послідовного" алгоритмів дає хороші результати для паралельної системи, але воно не має практичного глузду!

Вправи і завдання до теми №2

1. Паралельна програма виконується на MIMD – системі з 100 процесорами, 3% всіх команд при проході програми виконуються послідовно, а решту може виконуватись паралельно на всіх процесорах. Яке значення має показник прискорення цієї програми на даній програмі?

2. Деяка паралельна програма, що має 10% послідовну частину, має виконуватись на MIMD – системі. Чи існує деякий максимально можливий показник прискорення, незалежний від кількості процесорів системи?

3. Паралельна програма має виконуватись на MIMD – системі з 100 процесорами, проте:
   - 2% всіх команд при проході програми мають виконуватися послідовно;
   - 20% всіх команд можуть виконуватись тільки на 50 процесорах.

Яке значення має показник прискорення для цієї програми?

4. Паралельна програма має виконуватись на SIMD-системі, що має 10000 процесорних елементів, однак вона містить при виконанні 20% скалярних команд. Решту – векторні команди, що виконуються на всіх ПЕ. Яке значення має показник прискорення для цієї програми?

5. Паралельна програма має виконуватись на SIMD-системі, що має 10000 процесорних елементів. Якщо всі ПЕ були активними впродовж 30% загальної тривалості виконання програми, а решту часу були неактивними, яке значення має показник прискорення для цієї програми?

6. Паралельна програма має виконуватись на SIMD-системі, що має 100 000 процесорних елементів, проте:

- 20% всіх виконуваних інструкцій є скалярними командами;
- 10% всіх інструкцій можуть виконуватись векторно тільки на 100 процесорах;
- 40% всіх інструкцій можуть виконуватись векторно на 50 000 процесорах;
- решту інструкцій можуть виконуватись векторно на всіх процесорах.

Яке значення має показник прискорення для цієї програми?

7. Чи можна для оцінки продуктивності процесора з рухомою крапкою використовувати одиницю вимірювання MIPS?

8. Чому для оцінки продуктивності паралельних систем неефективний метод обчислення продуктивності складових частин?

9. Які недоліки стосовно обчислення продуктивності має закон Амдала?

10. Як впливають параметри комунікацій на загальну продуктивність систем різного типу?