Національний технічний університет України
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Дослідження і проектування комп'ютерних систем

КОНСПЕКТ ЛЕКЦІЙ
ЗАВДАННЯ НА КУРСОВУ РОБОТУ
до студенів спеціальності 123 - Комп'ютерна інженерія

Рекомендовано Вченою радою факультету інформатики та обчислювальної техніки

Київ КПІ ім. Ігоря Сікорського

2022

Розділ 1.

1.1. Складність алгоритмів

Існує декілька критеріїв оцінки складності алгоритмів. Однак найбільш часто в якості таких критеріїв використовується показник зростання тимчасових витрат та/або обсягів пам'яті, які необхідні для вирішення завдань при збільшенні обсягів вхідних даних.

Для цієї мети з кожним завданням зв'язується деяке число, що характеризує обсяг вхідних даних. Таке числа називається розміром завдання. При цьому істотне значення має показник складності алгоритму при граничних значеннях вхідних даних. Зазвичай цей показник називається асимптотичною тимчасовою (ємкісною) складністю. Цей показник визначає максимальну розмірність входу завдання (алгоритму), яка може бути вирішена за розумний час. [1]

Показники ефективності алгоритмів

Алгоритми, розроблені для вирішення однієї і тієї ж задачі, можуть значно відрізнятися по ефективності. Тому для характеристики їх якості вводять показники ефективності. Розглянемо найбільш поширені з них.

1. Кількість операцій - часова ефективність (time efficiency), показує наскільки швидко працює алгоритм.
2. Обсяг споживаної пам'яті - просторова ефективність (space efficiency), відображає максимальну кількість пам'яті, необхідної для виконання алгоритму.

Існують і інші показники, які має сенс розглядати, якщо вони в значній мірі впливають на процес вирішення завдання. Наприклад, для алгоритмів, що працюють з даними на зовнішніх носіях інформації (жорсткі диски, мережеві сховища), доцільно враховувати кількість звернень до зовнішньої пам'яті, а в алгоритмах, що використовують мережеві канали зв'язку, важливо брати до уваги кількість переданих повідомлень (мережевих пакетів) [2].

Введення показників ефективності дозволяє проводити аналіз алгоритмів з метою порівняння їх між собою і оцінювання потреби того чи іншого алгоритму в обчислювальних ресурсах: процесорному часі, пам'яті, пропускній здатності мережі [2].

Підрахунок числа операцій алгоритму

Для більшості алгоритмів кількість виконуваних ними операцій безпосередньо залежить від розміру вхідних даних. Чим більше вхідних даних, тим довше працює алгоритм. Кількість операцій алгоритму можна виразити як функцію від одного або декількох параметрів, пов'язаних з розміром вхідних даних.

RAM-машина. Для підрахунку числа операцій, що виконуються алгоритмом, необхідно формально описати систему команд деякого обчислювача. В якості такого виконавця будемо використовувати модель однопроцесорної обчислювальної машини з довільним доступом до пам'яті (Random Access Machine - RAM) (2). Домовимося, що машина має необмежену пам'ять і функціонує за такими правилами:

- Для виконання арифметичних і логічних операцій (+, -, *, /,%) потрібно один часовий крок (такт процесора);

- Кожне звернення до осередку в оперативній пам'яті для читання або запису займає один часовий крок;

- Виконання умовного переходу (if -$then$-$else$) вимагає обчислення логічного вираження і виконання однієї з гілок if -$then$-$else$;

- Виконання циклу (for, $while$, do) має на увазі виконання всіх його ітерацій, в свою чергу, виконання кожної ітерації вимагає обчислення умови завершення циклу і виконання його тіла.

Приклад. Алгоритм обчислення чисел Фібоначі на мові python.
Введемо позначення: n - вхідні дані (розмірність завдання), c - константа.

```
import sys
n = int(sys.argv[1])
a=0
b=1
s=0
time=0
for count in range(n):
        time=time+1 # + одна операція
        s=a+b
        time=time+1 # + одна операція
        print count, ' ' ,s
        a=b
        time=time+1 # + одна операція
        b=s
        time=time+1 # + одна операція
print t/n # відповідь 4
# Складність даного алгоритму пропорційна 4*n або const*n. Якщо відкинути константу, то складність пропорційна n.
```

Найбільш уживані класи складності алгоритмів представлені в таблиці 1.1.

Таблиця 1.1. Класи складності алгоритмів (3)

позначення класу складності	Назва класу	Приклад
O (1)	Константна складність	Алгоритм визначення парності цілого числа. Час виконання таких алгоритмів (або обсяг споживаної пам'яті) не залежить від розміру вхідних даних.
O (log n)	Логарифмічна складність	Алгоритм бінарного пошуку в упорядкованому масиві. Такі алгоритми на кожному кроці обробляють лише частину вхідного набору даних.
O (n)	Лінійна складність	Алгоритм пошуку мінімального елемента в неврегульованому масиві. Алгоритми з такою складністю виконують перегляд всього набору вхідних даних.
O (n log n)	Лінійно-логарифмічна складність	Алгоритм сортування злиттям. Така складність характерна для алгоритмів, розроблених з використанням методу декомпозиції (розділяй і володарюй).
O (n^2)	Квадратична складність	Алгоритм сортування вибором.
O (n^3)	Кубічна складність	Алгоритм множення квадратних матриць за визначенням.
O (2^n)	Експоненціальна складність	Алгоритми, що оброблюють всі підмножини деякої безлічі з n елементів.
O ($n!$)	Факторіальною складність	Оптимізаційні алгоритми, реалізують повний перебір безлічі допустимих рішень завдання.

З огляду на колосальне зростання продуктивності обчислювальних систем нинішнього покоління, часто робиться висновок, що ефективність застосовуваних алгоритмів в даний час не має ніякого значення. Однак, цей висновок є неправильним, так як збільшення продуктивності дозволяє вирішувати все більш великі завдання, складність яких часто збільшуються швидше, ніж продуктивність систем.

У таблиці 1.2. представлено тимчасову складність п'яти алгоритмів обробки входу розміру n, де одиницею часу є одна мілісекунда [1]. Тоді алгоритм A_1 може обробляти за одну секунду вхід розміру 1000, у той час, як алгоритм A_5 - вхід розміру 9. Таким чином, відмінність в розмірності за 1с досягає 3 порядків, а за 1 годину - 6 порядків.

У таблиці 1.3. розглядається ефект збільшення розмірності при збільшенні продуктивності на 1 порядок. З таблиці можна зробити висновок, що результати такого збільшення продуктивності дають дуже скромні результати в плані збільшення розміру завдання. Так, якщо для алгоритму A_3 розмір завдання потроюється, то для алгоритму A_5 десятикратне збільшення продуктивності збільшує розмір завдання, яке можна вирішити, тільки на три.

Таким чином, якщо для обчислювальних систем з послідовною організацією обчислень ефективність рішень визначається тільки алгоритмом, то в обчислювальних системах з паралельною організацією обчислень істотне значення набуває архітектура системи і кількість процесорів (комп'ютерів).

Тому в подальшому, під обчислювальної системою будемо розуміти сукупність алгоритмів і архітектури [1 база].

Табл. 1.2. Граничні розміри задач, які визначаються швидкістю росту складності (2)

Алгоритм	Часова складність	Максимальний розмір задачі		
		1 с.	1 хв.	1 год.
A_1	n	1000	$6*10^4$	$3,6*10^6$
A_2	$n\log_2 n$	140	4893	$2*10^5$
A_3	n^2	31	244	1897
A_4	n^3	10	39	153
A_5	2^n	9	15	21

Табл. 1.3. Ефект від прискорення КС у 10 разів (2)

Алгоритм	Часова складність	Максимальний розмір задачі	
		До прискорення	Після прискорення
A_1	n	S_1	$10\ S_1$
A_2	$n\log_2 n$	S_2	$\approx 10\ S_2$
A_3	n^2	S_3	$3,165*\ S_3$
A_4	n^3	S_4	$2,15*\ S_4$
A_5	2^n	S_5	$S_5 + 3,3$

В літературі визначаються наступні випадки роботи алгоритмів: [1,2]

- Кращий випадок (best case) - набір вхідних даних, на якому алгоритм виконується за найменше число операцій.
- Найгірший випадок (worst case) - примірник завдання, на якому алгоритм виконується за найбільшу кількість операцій. Час роботи алгоритму в гіршому випадку - теоретична верхня межа часу його роботи.

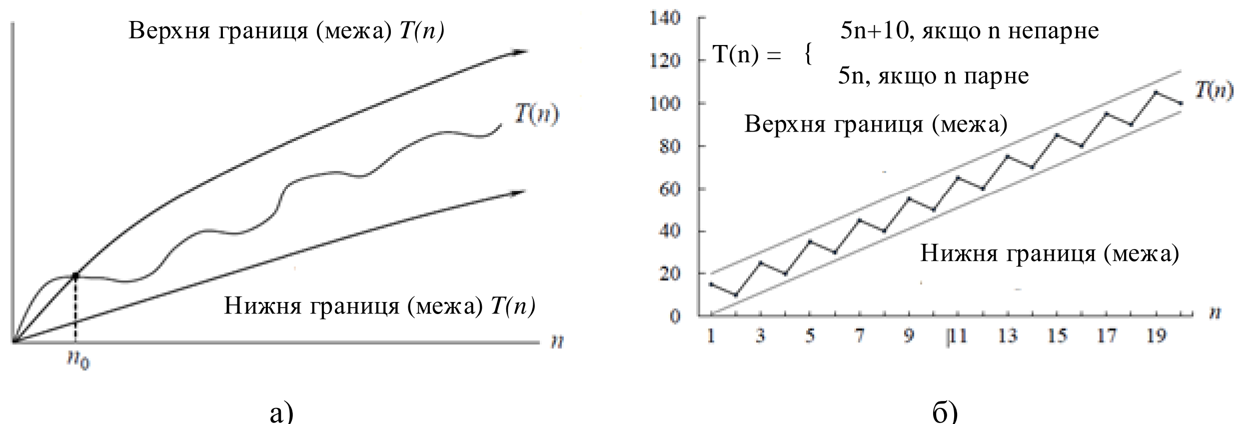
- Середній випадок (average case) - «середній» екземпляр завдання, при якому оцінюється математичне сподівання кількості операцій, які виконуються алгоритмом.

Під час аналізу алгоритму будемо приділяти увагу найгіршому часу роботи, тобто максимальному часу для будь-яких вхідних даних.

Швидкість зростання функцій. Асимптотичні позначення.

Час виконання алгоритму в гіршому, середньому і кращому випадках можна уявити, як функцію $T\ (n)$ від розміру його вхідних даних. Однак аналізувати ефективність алгоритмів за такими функціями досить важко з ряду причин. Як правило, функція $T\ (n)$ часу виконання алгоритму має велику кількість локальних екстремумів - нерівний графік з виступами і западинами (Рис. 1.1).

Наприклад, можлива ситуація, коли час виконання алгоритму на вхідних масивах з непарної довжиною буде більше часу виконання на масивах з парної довжиною. У цьому випадку графік $T\ (n)$ буде мати пилкоподібний вид, як на рисунку 1.1б. Тому набагато простіше працювати з верхньою і нижньою межами (оцінками) часу виконання алгоритму.



а) б)

Рис 1.1. Приклад можливих верхньої і нижньої границь (меж) часу $T\ (n)$ виконання алгоритмів [3].

Для позначення меж функцій $T\ (n)$ в теорії обчислювальної складності алгоритмів (computational complexity theory) використовують асимптотичні позначення: O (о велике), Ω (омега велике), Θ (тета велике), а також o (о мале) і ω (омега мале) (3).

Далі будемо вважати, що областю визначення функцій $f\ (n)$ і $g\ (n)$, які висловлюють число операцій алгоритму, є безліч невід'ємних цілих чисел ($n \in$ {0, 1, 2,...}). Функції $f(n)$ і $g(n)$ є асимптотично невід'ємними - при великих значеннях n вони приймають значення, які більші або рівні нулю.

O-позначення

O-позначення використовують, якщо необхідно вказати **асимптотичну верхню межу** (asymptotic upper bound) для функції $f\ (n)$, числа операцій алгоритму. Кажуть, що функція $f(n)$ належить множині функцій $O\ (g\ (n))$, що записується як $f\ (n) \in O\ (g\ (n))$, якщо існує позитивна константа $c > 0$ і $n_0 \in \{0, 1, 2, \ldots\}$, Такі що для будь-яких $n \geq n_0$

$$0 \leq f\ (n) \leq cg\ (n).$$

Формально множина $O\ (g\ (n))$ обирається наступним чином:

$$O\left(g\ (n)\right) = \{f(n): \exists_c > 0,\ \ n_0 \in \{0,1,2,..\}, \text{такі, що } 0 \leq f(n) \leq cg(n), \forall n \geq\ n_0\}$$

Інакше кажучи, $O\ (g\ (n))$ - це множина всіх функцій, значення яких при великих n не перевищують значення $cg\ (n)$.

Зазвичай факт належності функції $f\ (n)$ безлічі $O\ (g\ (n))$ записують як $f\ (n) = O\ (g\ (n))$.

Для доведення $f(n) = O(g(n))$ потрібно знайти константи $c > 0$ і $n_0 \in \{0, 1, 2, \ldots\}$, які забезпечують виконання нерівності (1.2). Якщо доведено, що $f(n) = O(g(n))$, то кажуть, що функція асимптотично обмежена зверху функцією g із точністю до постійного множника. Таким чином, добуток $cg(n)$ є асимптотичною оцінкою зверху часу $f(n)$ роботи алгоритму. Читається як «f від n є O велике від g від n».

Ω-позначення

Ω-позначення використовується для запису асимптотично нижньої межі (asymptotic lower bound) для функції $f(n)$. Кажуть, функція $f(n)$ належить множині $\Omega(g(n))$ (записується як $f(n) \in \Omega(g(n))$). Читається як «f від n є омега велике від g від n».

Θ-позначення

Θ-позначення дозволяє записати **асимптотично точну оцінку** (Asymptotic tight bound) для функції $f(n)$. Функція $f(n)$ належить безлічі функцій $\Theta(g(n))$, записується як $f(n) \in \Theta(g(n))$. Читається як «f від n є тета велике від g від n».

Висновки

Якщо для машини з послідовною організацією обчислень час вирішення завдання залежить тільки від складності алгоритму, то для машин з паралельною організацією обчислень час вирішення залежить як від складності алгоритму, так і від архітектури системи (числа процесорів).

Під обчислювальною системою будемо розуміти поєднання архітектури і алгоритму. Наше завдання полягає в тому, щоб знайти такі архітектури обчислювальних систем, які забезпечать лінійну залежність продуктивності від числа процесорних елементів.
З цією метою, на підставі зміни зернистості паралельної обробки варіюватимемо і цією залежністю, наближаючи її до лінійної.

Завдання:

Заповніть нижче наведену таблицю, рядки якої відповідають різним функціям f (n), а стовпці - значенням часу t (1). Визначте максимальні значення n, для яких задача може бути вирішена за час t, якщо передбачається, що час роботи алгоритму, необхідний для вирішення завдання f (n) секунд.

Таблиця 1.4. Залежність часу від розміру задачі.

	Секунда	Минута	Час	День	Месяц	Год	Век
$\lg n$							
$\sqrt{n}$							
n							
$n \lg n$							
n^2							
n^3							
2^n							
$n!$							

References
2. **АХО, А. Дж. Хопкрофт, Дж. Ульман. Построение и анализ вычислительных алгоритмов. Москва, 1979.**

3. **Курносов М.Г. Введение в структуры и алгоритмы обработки данных. – Новосибирск: Автограф, 2015. – 179 с.**

Метрика паралельних обчислень

Метрика паралельних обчислень - це система показників, яка дозволяє оцінювати переваги, отримані при паралельному вирішенні задачі на N процесорах, в порівнянні з послідовним рішенням тієї ж задачі на єдиному процесорі. В якості базових показників для виділення метрик виділимо наступні:

$\boldsymbol{N}$ - кількість пристроїв (процесорів, комп'ютерів), які використовуються для реалізації паралельних обчислень;

$\boldsymbol{O(1)}$ - обсяг обчислень, представлений кількістю операцій, виконуваних по ходу послідовних обчислень на 1 процесорі;

$\boldsymbol{O(N)}$ - обсяг обчислень, представлений кількістю операцій, виконуваних по ходу паралельних обчислень на N процесорах;

$\boldsymbol{T_S}$ або $\boldsymbol{T(1)}$ - час виконання при послідовному виконанні задачі;

$\boldsymbol{T_P}$ або $\boldsymbol{T(N)}$ - час виконання при паралельному виконанні задачі;

Нехай середній час виконання однієї операції дорівнюватиме деякій умовній одиниці часу. Тоді

$$\boldsymbol{T_S = O(1)} \quad (1.1)$$

або

$$\boldsymbol{T(1) = O(1),}$$

$$\boldsymbol{T_P < O(N)} \quad (1.2)$$

або

$$\boldsymbol{T(N) < O(N).}$$

Звідси випливає, що прискорення обчислювального процесу при вирішенні задач паралельним способом дорівнюватиме

$$\boldsymbol{S(N) = \frac{T(1)}{T(N)}} \quad (1.3)$$

або

$$\boldsymbol{S(N) = \frac{T_S}{T_P}}$$

Тоді ефективність паралельної обробки буде дорівнює

$$\boldsymbol{E(N) = \frac{S(N)}{N}} \quad (1.4)$$

а вартість обчислювального процесу буде дорівнювати

$$\mathrm{C}(\boldsymbol{1}) = \boldsymbol{T_S \cdot 1 = T(1) \cdot 1} \quad (1.5)$$

$$\mathrm{C}(\boldsymbol{N}) = \boldsymbol{T_S \times N = T(N) \times N.} \quad (1.6)$$

Для вирішення питань розподілу і призначення окремих ділянок задачі, яка розв'язується, між процесорами та для реалізації процесів взаємодії і синхронізації процесів, а також інших питань паралельної обробки інформації, з'являється необхідність у введенні додаткових операцій, в порівнянні з тими, які використовувалися при послідовному способі їх обробки. Сукупність цих додаткових операцій і, відповідно,

пов'язані з ними додаткові обсяги обчислювальних і організаційних робіт будемо надалі називати накладними витратами паралелізму $\Delta(N)$, де

$$\Delta\ (N) = C(N) - C(1). \tag{1.7}$$

Накладні витрати паралелізму і різні можливості завдань з точки зору їх розпаралелювання істотно збільшують витрати часу. Так при використанні N-процесорної або N-комп'ютерної системи ми вправі очікувати N-кратного зменшення тимчасових витрат або, іншими словами, N-кратного збільшення продуктивності. Але, як правило, ефект від використання паралельних систем виявляється набагато скромніше.

Першою оцінкою ефективності паралельної обробки була гіпотеза Мінського, відповідно до якої продуктивність паралельних систем зі збільшенням числа процесорів зростає за логарифмічним законом. Ця гіпотеза по суті виключала розрахунки на широке використання паралелізму в плані кардинального підвищення продуктивності обчислювальних систем. На щастя Мінський зробив цей висновок на основі дослідження проблемно орієнтованої системи, а саме, матричної системи ***ILLIAC-4***. У паралельних системах широкого призначення оцінки ефективності паралельної обробки є більш високими, проте в будь-якому випадку залежність продуктивності паралельних систем від числа процесорів є нелінійною.

Основним предметом дослідження в даному курсі є масштабовані системи, тобто системи з практично необмеженими можливостями нарощування ресурсів, орієнтовані на масове розпаралелювання. Основним завданням таких систем є лінеаризація залежності продуктивності паралельних систем від числа процесорів. З цією метою вираз (1.7) представимо в такий спосіб:

$$\Delta\ (N) = N \times T(N) - 1 \times T(1). \tag{1.8}$$

З цього виразу випливає:

$$T(N) = \frac{\Delta(N)+T(1)}{N}.$$

Тоді прискорення

$$S(N) = \frac{T(1)\times N}{\Delta\ (N)+T(1)},$$

а ефективність паралельної обробки

$$E(N) = \frac{T(1)}{\Delta\ (N)+T(1)} \tag{1.9}$$

або

$$E(N) = \frac{O(1)}{\Delta\ (N)+O(1)}\ . \tag{1.10}$$

З виразу (1.10) випливає

$$O(1) = \frac{\Delta\ (N) \times E(N)}{1-E(N))}. \tag{1.11}$$

Вираз (1.11) пов'язує між собою ефективність паралельної обробки і обсяг обчислень, який представлений кількістю операцій, що виконуються під час вирішення завдання, і який безпосередньо пов'язаний з розмірністю завдання. Таким чином, варіюючи розмірністю завдання, можна отримувати необхідні показники ефективності при вирішенні відповідних завдань, а отже, і вирішувати найважливішу проблему паралелізму, яка пов'язана з нелінійним характером збільшення продуктивності зі збільшенням числа процесорів. Так як досягнення лінійного характеру цієї залежності передбачає, що показник ефективності паралельних обчислень повинен бути величиною постійною, то на основі (1.11) можна записати:

$$\boldsymbol{O(1) = n = k \times \Delta\,(N)}, \tag{1.12}$$

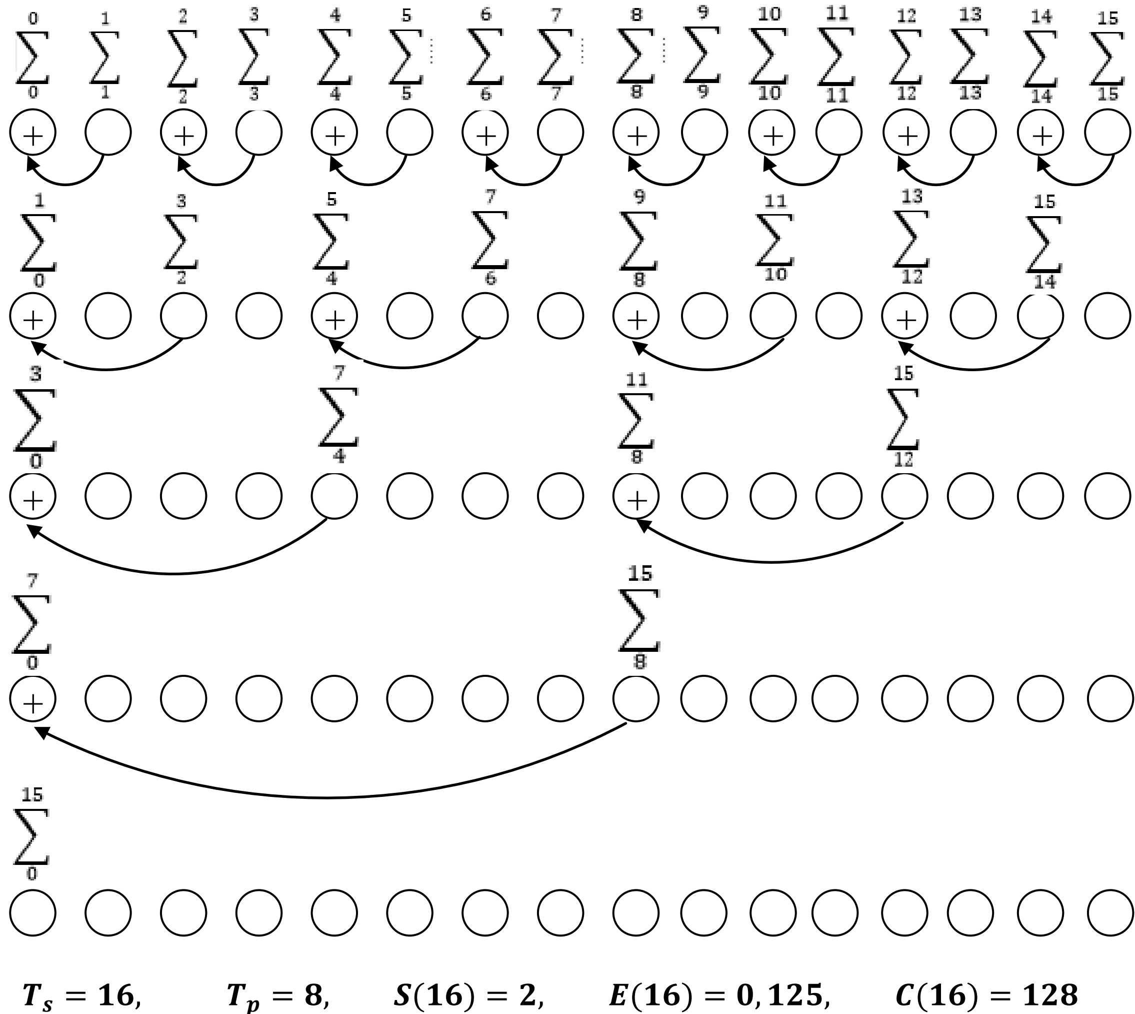
де $\boldsymbol{n}$ - розмірність задачі. а $\boldsymbol{k}$ - коефіцієнт пропорційності, що дорівнює

$$\boldsymbol{k = \frac{E(N)}{1 - E(N)}}$$

Таким чином, при збільшенні розмірності задачі пропорційно зростанню накладних витрат з'являються передумови для лінеаризації залежності продуктивності від числа процесорів.

1.2. Ізоефективні системи

Розглянемо найпростіший, але досить типовий приклад нелінійного характеру нарощування продуктивності обчислювальних систем при збільшенні числа процесорів або комп'ютерів. У цьому прикладі необхідно знайти суму 16 чисел на 16 процесорах. На рис. 1.3 представлена послідовність виконання цієї операції, де дуги визначають пересилання даних з одного процесора в інший, а знак суми - підсумовування суми в приймаючому процесорі з сумою передавального процесора. При цьому передбачається, що час операції пересилання і час операції підсумовування виконуються за один і той же проміжок часу.



$$T_s = 16, \quad T_p = 8, \quad S(16) = 2, \quad E(16) = 0,125, \quad C(16) = 128$$

Рис. 1.3. Послідовне виконання операцій додавання для N = 16, n = 16

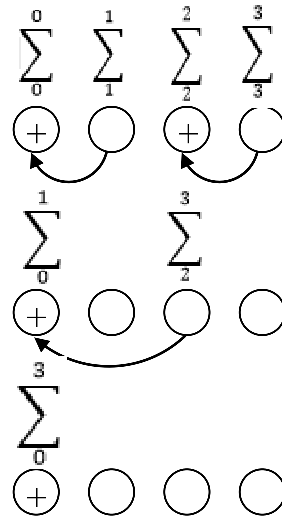
Таким чином, 16-кратне збільшення числа процесорів дозволяє тільки в два рази зменшити час виконання операції, тобто тільки в два рази прискорити обчислювальний процес. При цьому ефективність паралельної обробки дорівнюватиме $\boldsymbol{E(N)}$ **=0,125.**

Для випадку, коли **n = 4** і **N = 4**, метричні характеристики будуть наступними (рис. 1.4):

$$T_s = T(1) \approx 4; \quad T_P = T(N) \approx 4; \quad S(4) = 1; \quad E(4) = 0.25; \quad C(4) = 16.$$

Це означає, що 4-кратне збільшення числа процесорів в даному прикладі не приносить ніякої користі в плані підвищення продуктивності. Але, якщо в останньому випадку збільшити розмірність завдання, скажімо, до 16, то відповідно до (1.12) ми

повинні отримати певний ефект. Розглянемо два алгоритми виконання цієї процедури (рис.1.5, 1.6). В якості топологічної організації системи виберемо гіперкуб другого порядку. Перший алгоритм представлено на рис. 1.5а, 1.5б, а другий на рис. 1.6а, 1.6б.



$\boldsymbol{T_s = 4}$, $\boldsymbol{T_p = 4}$, $\boldsymbol{S\,(4) = 1}$, $\boldsymbol{E\,(4) = 0,25}$, $\boldsymbol{C\,(4) = 16}$

Рис. 1.4. Послідовність виконання операцій додавання для N = 4, n = 4.

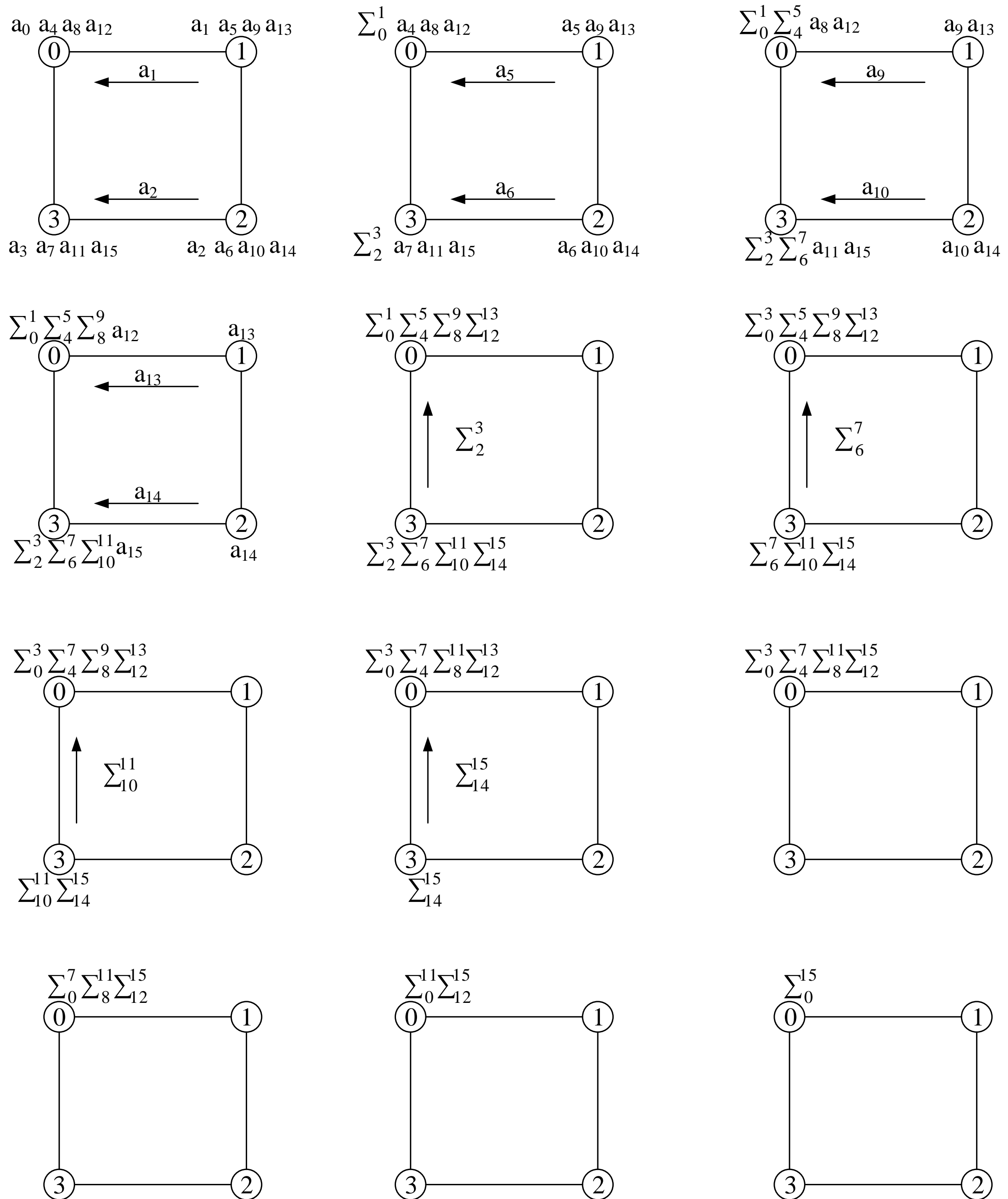
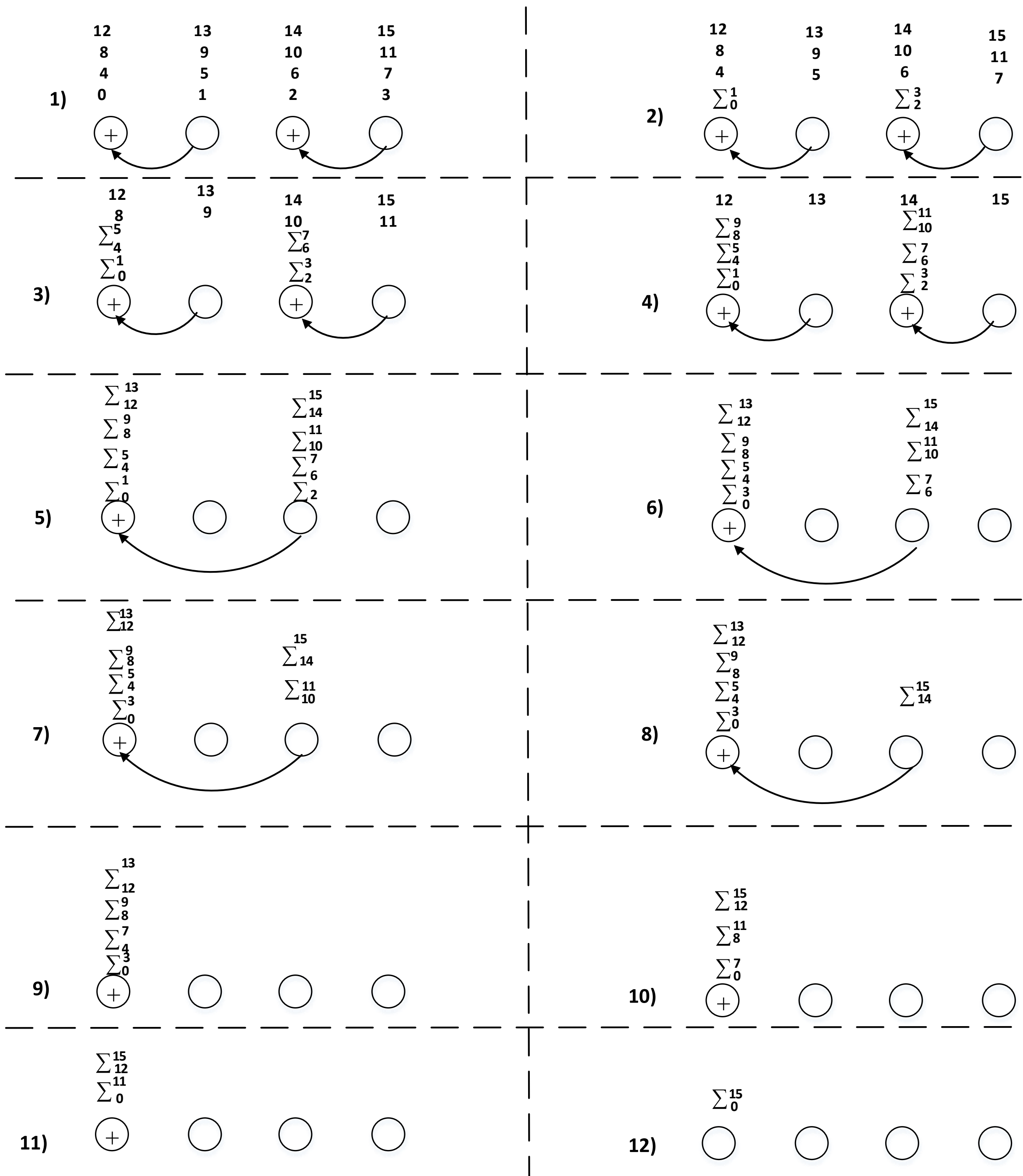


Рис. 1.5а. Послідовність виконання операцій додавання для N=4, n=16.

Алгоритм, що представлений на рис. 1.5а, є явно неефективним, так як при його використанні прискорення паралельної обробки виявляється за величиною менше одиниці.



$$T_p = 2\frac{n}{N}log_2\mathrm{N} + \frac{n}{N} - 1 \approx 2\frac{n}{N}log_2N + \frac{n}{N} = 20$$

$$S = \frac{nN}{2nlog_2\mathrm{N} + n} = \frac{16}{20} = \frac{4}{5}$$

$$E = \frac{S}{\mathrm{N}} = \frac{n}{2nlog_2\mathrm{N} + n} = \frac{1}{5}$$

$$C_p = N \cdot T_p = 4 \times 20 = 80$$

Рис. 1.5б. Послідовність виконання операцій додавання для N=4, n=16.

Другий алгоритм, що представлений на рис. 1.6б, в плані прискорення і ефективності дає досить хороші результати.

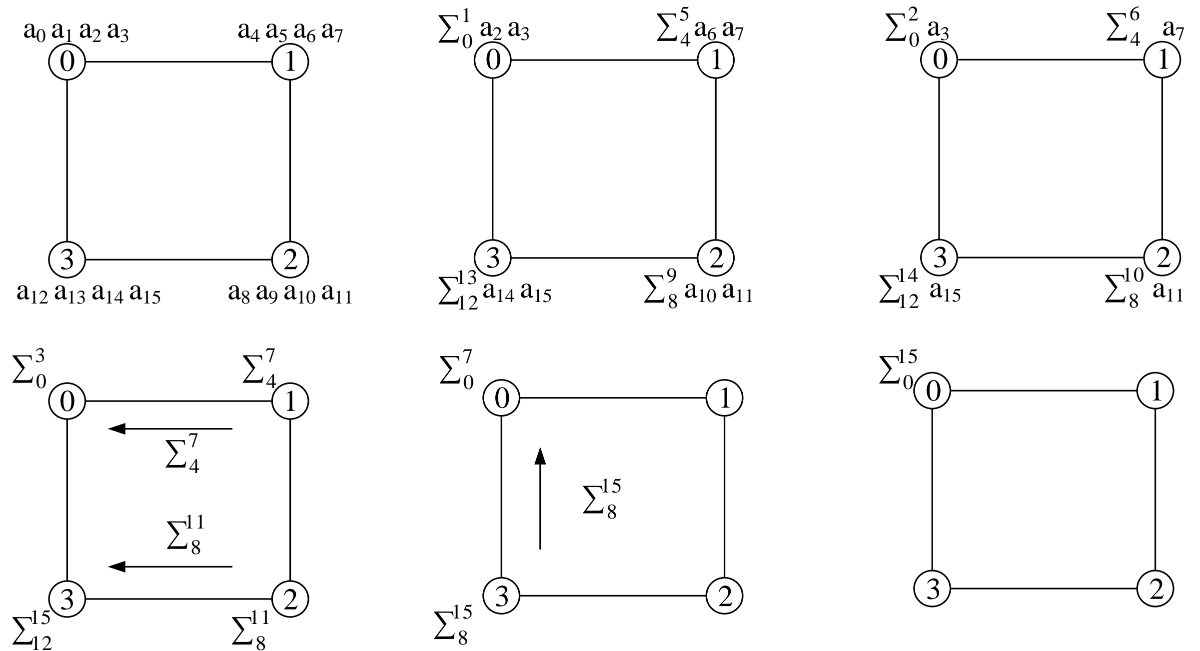
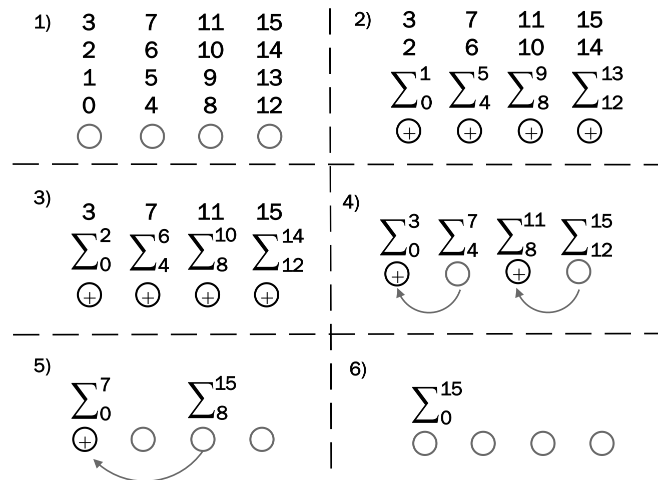


Рис. 1.6а. Послідовність виконання операцій додавання для N=4, n=16.



$$T_p = 2log_2\mathrm{N} + \frac{n}{N} - 1 \approx 2log_2N + \frac{n}{N} = 8$$

$$S = \frac{nN}{2nlog_2\mathrm{N} + n} = \frac{16}{8} = 2$$

$$\boldsymbol{E = \frac{S}{\mathrm{N}} = \frac{1}{1 + 2\frac{N}{n}log_2\mathrm{N}} = \frac{1}{2}}$$

$$C_p = N \times Tp = 4 \times 8 = 32$$

Рис. 1.6б. Послідовність виконання операцій додавання для N=4, n=16.

Крім того, особливий інтерес тут представляє формула ефективності паралельної обробки:

$$E = \frac{S}{\mathrm{N}} = \frac{1}{1+2\frac{N}{n}log_2\mathrm{N}} \tag{1.13}$$

яка є окремим випадком формули (1.12) і визначає можливість досягнення необхідної ефективності паралельних обчислювальних систем. Така можливість надається за рахунок варіювання параметрами n і N. На основі такого варіювання можна досягти лінійного нарощування продуктивності при збільшенні кількості процесорів. Це означає, що при виконанні обчислювальних процесів можна заздалегідь визначати необхідну ефективність їх реалізації.

Такі системи, в яких ефективність вирішення завдання може задаватися заздалегідь і постійно підтримуватися, називають ізоефективними системами.

Приклад. Нехай є система, що включає в себе 4 процесора. Потрібно знайти розмірність n задачі сумування (додавання) для випадку ізоефективності E = 0.8.

Таким чином, E = 0.8 і N = 4. Виходячи з формули (1.13) отримаємо:

$$0{,}8 = \frac{1}{1 + 2\frac{4}{n}log_2 4} = \frac{1}{1 + \frac{16}{n}}$$

$$\frac{12{,}8}{n} = 0.2$$

$$n = 64$$

Таблиця 1.4. Розмірність n задачі як функція від кількості процесорів N при ефективності E = 0.8

n	N=1	N=4	N=8	N=16
64	1	0.8		
192	1		0.8	
512	1			0.8

1.3. Ефективність паралельної обробки

Ефективність паралельної обробки тісно пов'язана з таким її показником, як ступінь паралелізму. При цьому будемо розрізняти ступінь паралелізму ***L(t)***, яку можна визначити числом процесорів, які паралельно беруть участь при виконанні програми вирішення відповідної задачі в момент часу ***t***, максимальну ступінь паралелізму ***Lmax***, яка має місце при рішенні даної задачі, і середню ступінь паралелізму $\bar{L}$, яка має найбільш впливове значення при визначенні ефективності паралельної обробки. Для визначення значення середнього ступеня паралелізму розглянемо графічне представлення ступеня паралелізму ***L(t)***, яке зазвичай називають профілем паралельних програм.

Приклад подання ступеня паралелізму показаний на малюнку 1.7., де загальний обсяг обчислювальної роботи буде дорівнювати $W = \Delta\sum_{i=1}^{N} i \times t_i$, де Δ - продуктивність одного процесора.

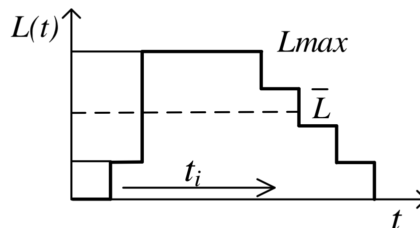


Рис 1.7. Профіль паралелізму програми

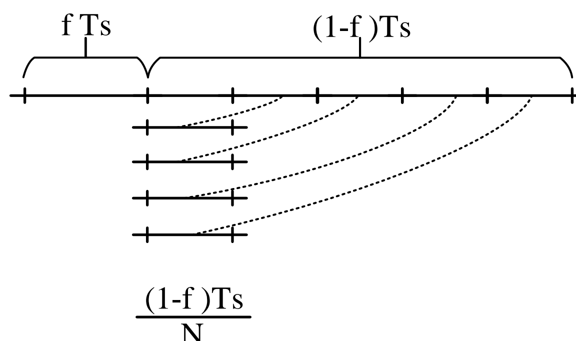
Середній паралелізм $\bar{L}$ визначається як:

$$\bar{L} = \frac{W}{\Delta \times \sum_{i=1}^{N} t_i}$$

Середній паралелізм зазвичай поступається за величиною значенню максимального ступеня паралелізму ***Lmax***. Таким чином $\bar{L} \leq Lmax$. Крім того, як правило, в кожній задачі або у відповідній їй програмі, мають місце фрагменти, які не піддаються розпаралелюванню. Один з розробників відомої системи IBM 360 Джон Амдал сформулював закон, який враховує наявніть в кожній програмі ділянок, що не піддаються розпаралелюванню.

Закон Амдала.

Розділимо загальний час Ts послідовного вирішення завдання на дві частини: послідовну $f \times Ts$ (яка не піддається розпаралелюванню) і паралельну частина $(1 - f) \times Ts$, (яка розв'язується паралельно), де f - частка послідовної частини програми. Паралельну частина потім розділимо на N фрагментів, які будуть виконуватися одночасно на N процесорах (рис. 1.8).



$\frac{(1-f)Ts}{N}$

Рис. 1.8. Ілюстрація закону Амдала

$$S = \frac{T_s}{T_p} = \frac{T_s}{\frac{(1-f)T_s}{N} + fT_s} = \frac{N}{Nf + 1 - f} = \frac{N}{1 + f(N-1)} \approx \frac{N}{f(N-1)} \approx \frac{1}{f} \qquad (1.14)$$

Таким чином, закон Амдала показує, що для завдання з $f \neq 0$ приріст ефективності при паралельній обробці залежить від алгоритму розв'язання задачі і істотно обмежений зверху. Дійсно, якщо послідовна частина програми становить хоча б 5%, то це означає, що прискорення паралельної обробки не може перевищувати 20-кратної величини. Про яке масове розпаралелювання може тоді йти мова? І тут у відповідь Амдалу дослідник з NASA Ames Research Джон Густафсон формулює новий закон.

Закон Густафсона.

Цей закон базується на дослідженні Густафсона, яке показало, що при збільшенні розмірності задачі зростає в основному паралельна частина програми, а послідовна частина за об'ємом обчислень залишається практично незмінною.

Тоді формула (1.14) прийме наступний вигляд:

$$S = \frac{fT_S + (1-f)T_S N}{fT_S + (1-f)T_S} = f + N - Nf = N + (1-N)f \approx N - Nf = N(1-f) \quad (1.15)$$

Таким чином, якщо частка послідовної частини програми є незначною, то ми отримуємо практично N-кратне прискорення. Ну і звичайно, зменшення значення f можна досягти шляхом підвищення розмірності задачі, що узгоджується з особливостями масового розпаралелювання.

Відповідно до цього закону, користувач не прагне до збільшення продуктивності або зменшення часу виконання завдання, а прагне до того, щоб час не збільшувався зі збільшенням розмірності задачі. При збільшенні розмірності задачі, послідовна частина залишається тією ж, а паралельна - збільшується, тим самим частка послідовної частини істотно зменшується.

Закон Ксіа-Хе Сан і Лайонел Наєм.

Узагальнення законів Амдала і Густафсона визначили Ксіа-Хе Сан і Лайонель Наєм. Мета такого узагальнення пов'язана з обсягами пам'яті, що не дозволяє нескінченно збільшувати розмірність задачі. З урахуванням обмежень на обсяги пам'яті вираз (1.16) запишемо в такий спосіб:

$$S = \frac{fT_S + G(N) \times (1-f)T_S N}{fT_S + \frac{G(N) \times (1-f)T_S}{N}} \quad (1.16)$$

де G (N) - допустима, в силу обмежень обсягів пам'яті, ступінь масштабування паралельної частини завдання. Тоді при G (N) = 1 вираз (1.16) набуде вигляду (1.14), що відповідає закону Амдала, а при G (N) = N - вид, що відповідає закону Густафсона (1.15).

Отримані формули не завжди точно визначають справжні показники прискорення паралельної обробки. Це пов'язано з тим, що вони не враховують витрат на вирішення питань взаємодії між процесорами паралельної системи.

На основі реальних можливостей розпаралелювання конкретних програм в реальних системах Алан Карп і Хорас Флетт запропонували використовувати еквівалент показника f, відомий як метрика Карпа-Флетта і позначається $\boldsymbol{e}$.

$$e = \frac{\frac{1}{S(N)} - \frac{1}{N}}{1 - \frac{1}{N}} \quad (1.17)$$

Чим менше значення $\boldsymbol{e}$, тим краще може бути розпаралелювання даної програми. Для завдань фіксованого розміру (як в постановці Амдала) ефективність паралельних обчислень зі збільшенням числа процесорів N зазвичай зменшується. За допомогою показника $\boldsymbol{e}$ можна оцінити причини зниження ефективності, обмежені можливостями розпаралелювання або комунікаційними витратами паралельних обчислень.

Розділ 2.

2.1. Топологічна організація масштабованих комп’ютерних систем

Найважливішим елементом архітектури масштабованих комп’ютерних систем є засоби обміну даними між комп’ютерами (процесорами). Організацію внутрішніх комунікаційних зв'язків системи зазвичай називають топологією системи, або топологічною організацією системи.

Розрізняють два типи топологій: статичні, при яких всі зєднання між компютерами (процесорами) фіксовані, і динамічні в яких між компютерами (процесорами) є перемикачі (комутатори). Системи з фіксованою топологією зазвичай називають безпосередньо-зв’язаними системи (БЗС) або системами «точка-точка»; системи з динамічними топологіями– комутованими системами.

2.1. Безпосередньо-зв’язані мережі

Масштабованість є найважливішим моментом про розробці мультикомпьютерних систем. Безпосередньо-зв’язані мережі (БЗМ) або мережі «точка-точка» є популярними мережевими архітектурами, які легко масштабуються. БЗМ складаються із набору вузлів, кожен з яких безпосередньо зв’язаний з підмережою других вузлів мережі. Загальним компонентом цих вузлів є router , який управляє передачею повідомлень між вузлами. З цієї причини БЗМ часто називають мережами, які засновані на маршрутизації. Кожен маршрутизатор (router) має безпосередні зв’язки з маршрутизатором сусідів. Зазвичай два сусідні вузли зв’язані між собою парою однонаправлених каналів в протилежних напрямках. Двунаправлений канал може бути використаний також для зв’язку сусідніх вузлів. Хоча функція маршрутизатора можуть бути виконана локальними процесором, для високопродуктивних систем використовуються виділені маршрутизатори. При цьому створюється передумова для сумісництва обчислювальних та комунікаційних операцій в кожному вузлі.

Кожен маршрутизатор підтримує деяку множину вхідних та вихідних каналів. Вбудовані канали або порти зв’язують локальний процесор/пам’ять з маршрутизатором. В закальному випадку необхідна пропускна здатність може забезпечити одна пара вбудованих каналів, але для виключення вузького місця між локальним процесором/пам’ять і маршрутизатором використовують більше вбудованих каналів. Зовнішні канали використовуються для взаємодії маршрутизаторів. На основі зв’язків вихідних каналів одного вузла з вихідними каналами других вузлів і формується мережа.

Безпосередньо зв'язані мережі (або мережі «точка-точка», або мережі з фіксованою топологією) є досить популярними, так як вони в повній мірі вирішують питання масштабованості систем. Спільним елементом в таких системах є роутер (або маршрутизатор), тому такі мережі називають мережі з маршрутизацією.

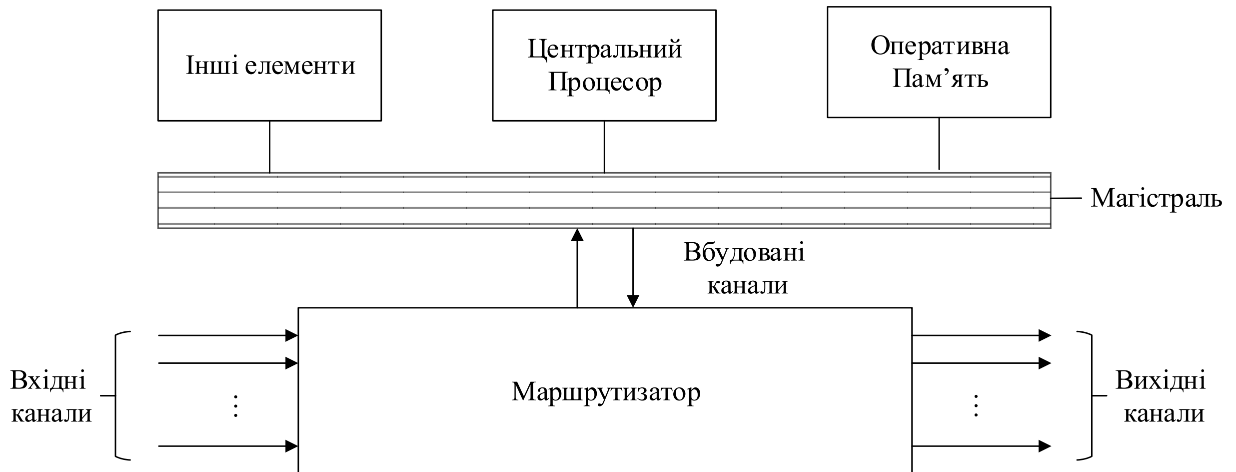


Рис. 2.1. Структура елемента безпосередньо зв'язаних систем

В комутованих мережах або мережах з реконфігурацією системи зв'язків, кожен вузол повинен мати відповідний адаптер мережі. За допомогою цього адаптера кожен вузол може підключатися до комутатора мережі. Кожен комутатор має множину портів за допомогою яких він може підключатися до іншого комутатора, до процесора, а також може залишатися відкритим, тобто не задіяним.

2.2. Комутовані мережі

Комутовані мережі – інший клас мережевих організацій. Взаємодія вузлів в даному випадку здійснюється на основі комутаторів. Кожен вузол в такій організації має мережевий адаптер, який підключається до мережевого комутатору. Кожен комутатор має множину портів. Кожен порт складається з одного вхідного і одного вихідного лінків. Множина портів кожного комутатора зв’язується з процесорами або залишаються відкритими, а інші порти зв’язуються з портами інших комутаторів для забезпечення зв’язності між процесорами. Зв’язки між комутаторами визначають реалізацію мережевої топології. При цьому у випадку регулярних зв’язків між комутаторами отримуємо регулярні топології, в іншому випадку – нерегулярні топології.

Комутовані мережі можна також моделювати графом *G(N,C)*, де *N* – множина комутаторів, а *C* – множина однонаправлених або двонаправлених лінків між комутаторами. Для аналізу властивостей таких мереж необхідно явно включити процесорні вузли в граф, при цьому потрібно враховувати, що кожен комутатор може підключатись до 0, одного або більше процесорів. Очевидно, що тільки ті комутатори, які підключені до декількох процесорів можуть бути джерелами або приймачами повідомлень. При передачі повідомлень від одного вузла іншому потрібно проходження лінку між джерелом і комутатором, а також лінків між останнім комутатором і приймачем. Таким чином, відстань між вузлами це відстань між комутаторами, які безпосередньо зв’язують ці вузли, плюс два елемента. Отже, діаметр – це відстань між двома комутаторами, які зв’язують ці вузли, плюс два елементи (рис. 2.2). У цьому випадку відстань між вузлами дорівнює числу комутаторів + 2 (кількість адаптерів) (рис. 2.3).

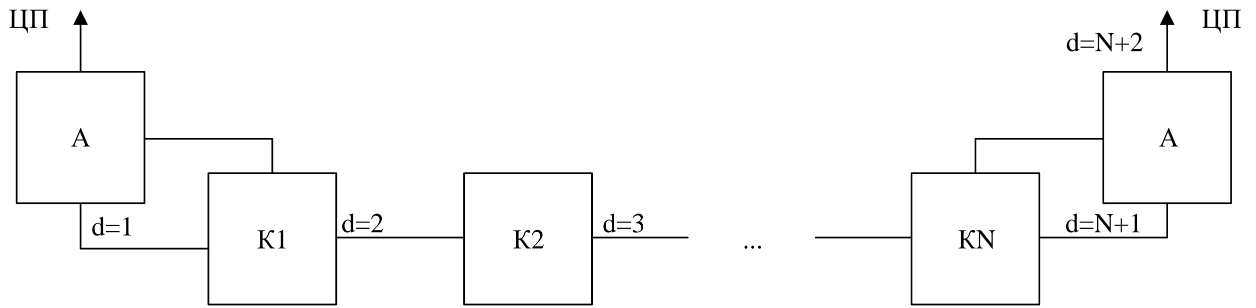


Рис 2.2. Відстань (d) між вузлами

Безпосередньо зв'язну мережу комутаторів можна розглядати як єдиний складений комутатор. Наприклад:

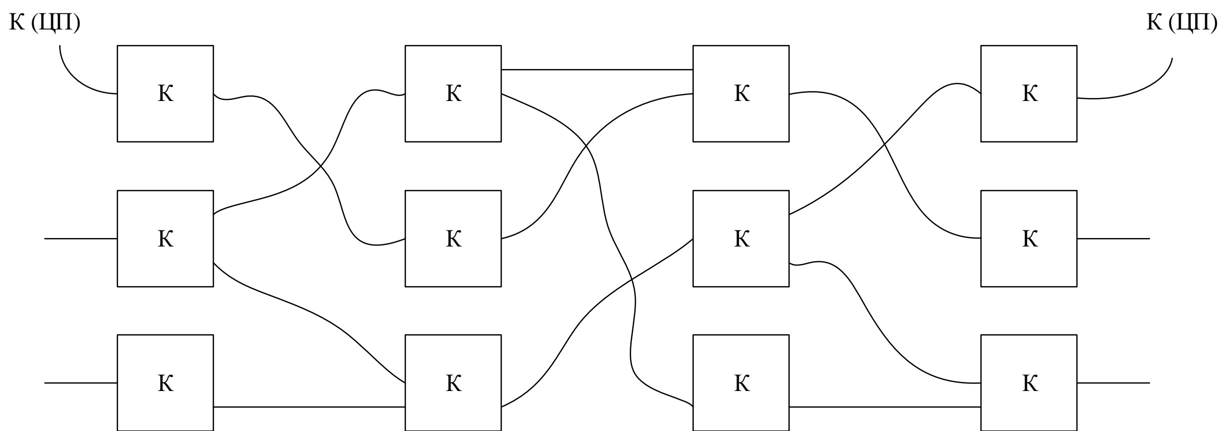


Рис. 2.3. Комутована мережа.

В якості елемента такого складеного комутатора (рис. 2.1) розглянемо схему представлену на рисунку 2.4.

Основні концепції

Рівні комутації можуть розрізнятись способом реалізації і відносним часом перебігу управляючих операторів, а також технікою перемикання. Крім цього, ці оператори можуть перекриватися в часі з процесами реалізації маршрутизації.

Управляючий потік – це протокол синхронізації для передачі і отримання одиниці інформації. Одиниця управляючого потоку – це така частина повідомлення, яка підлягає синхронізації. Ця одиниця визначається як найменша одиниця інформації, передача якої запитується відправником і підтверджується отримувачем. Запит/підтвердження сигналізують про успішну передачу й наявність буферного простору в приймачі.

Модель мережі і маршрутизатора

Архітектура узагальненого маршрутизатора має наступні головні компоненти:

1. Буфери: Це буфери типу FIFO для зберігання транзитних данних. Буфер асоціюється з кожним вхідним фізичним каналом і кожним вихідним фізичним каналом. В альтернативних розробках можуть мати місце тільки вхідні чи вихідні буфери.
2. Комутатор: Це елемент, який відповідає за зв'язування буферних входів маршрутизатора з вихідними буферами. Високо-швидкісний використовує повнозв'язний комутатор (crossbar), низько-швидкісний може бути різним.
3. Маршрутизація і арбітражний елемент: Цей компонент реалізує алгоритм маршрутизації, виділяє вихідний лінк для вхідного повідомлення у відповідності до набору комутаторів. Якщо декілька повідомлень одночасно запитують один і той самий

вихідний лінк, то даний компонент реалізує арбітраж цих запитів. Якщо запитуванні лінки зайняті, вхідні потоки залишаються в вхідних буферах.

4. Контролери лінків (LC): Потік повідомлень по фізичним каналам між сусідніми маршрутизаторами реалізується за допомогою LC.
5. Інтерфейс процесора: Цей компонент просто реалізує інтерфейс фізичного канала з процесором швидше, ніж з сусіднім маршрутизатором. Він складається з одного чи більше каналів впорскування в процесор. Канал впуску часто називається постачальником, а канал впорскування – каналом споживачем.

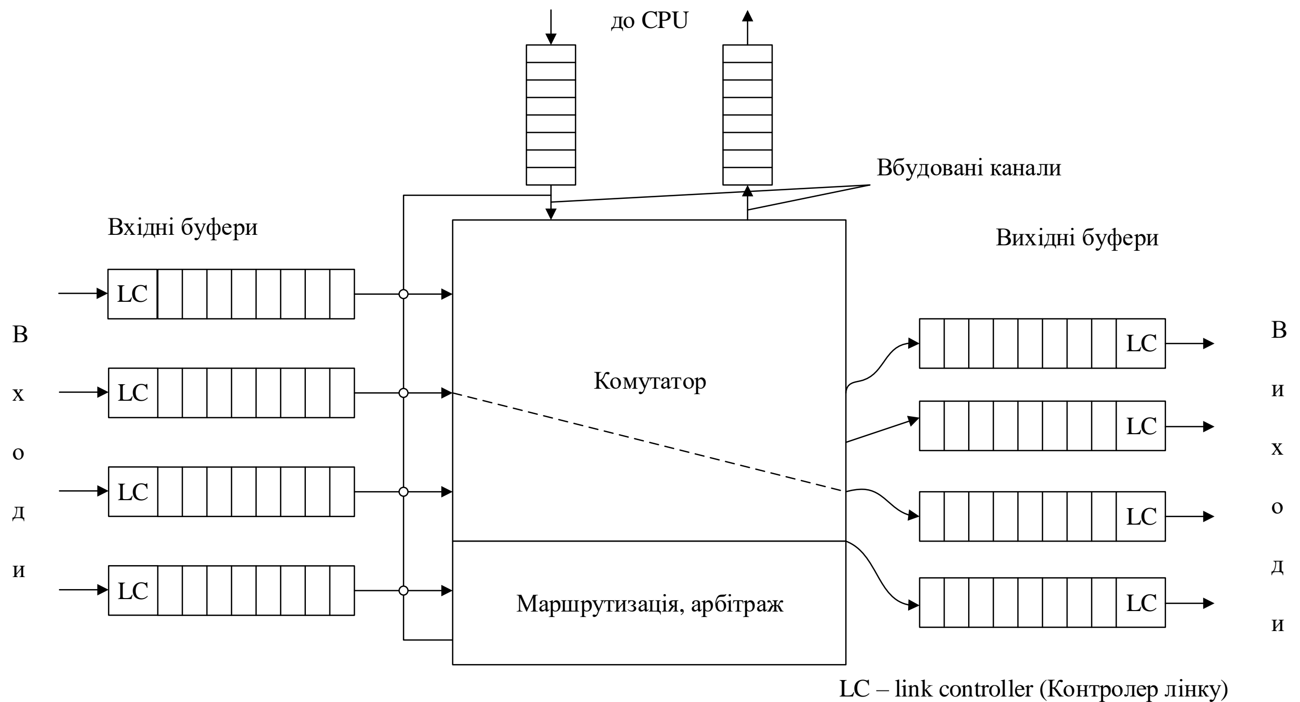


Рис. 2.4. Маршрутизатор комутованої мережі

При використанні такого маршрутизатора маршрутизація – це вибір вихідного буфера для кожного вхідного буфера. Арбітраж – вирішення конфліктної ситуації при рішенні задач маршрутизації. Вбудовані канали зв'язують комутатори з адаптерами центрального процесора.

У комутуючих мережах зазвичай розрізняють 3 рівня:
1) Фізичний.
2) Комутаційний.
3) Рівень маршрутизації.
На фізичному рівні вирішуються питання управління і синхронізації передачі інформації між сусідніми маршрутизаторами. Комутаційний рівень реалізує передачу інформації через мережу. На рівні маршрутизації вирішуються питання вибору вихідних буферів для кожного вхідного буфера, тобто визначається шлях передачі інформації.

Рівні комутації повідомлень

Багатопроцесорна комунікація може розглядатись як ієрархічна система обслуговування, яка починається з фізичного рівня, який призначений для синхронізації передачі бітових потоків, і закінчується високо-рівневими протоколами, де виконуються такі функції як формування пакетів, шифрування пакетів, стиснення даних і т.п.. Така система обслуговування характерна для локальних і глобальних мереж. Однак, ці реальні рішення не узгоджуються з набором рівнів для мультипроцесорних систем. При цьому передбачається

доцільним вирізняти три рівня операцій в мультикомп'ютерних мережах: рівень маршрутизації, рівень комутації, фізичний рівень.

Фізичний рівень зв'язується з протоколами лінкового рівня, які призначені для передачі повідомлень і іншої управляючої інформації між сусідніми маршрутизаторами. Рівень комутації використовує фізичні протоколи для реалізації механізмів переміщення повідомлень через мережу. Нарешті, протокол маршрутизації приймає рішення на предмет вибору кандидатів на вихідні канали в проміжних маршрутизаторах, а отже встановлює шляхи через мережу.

Тут розглянемо техніку реалізації в середі мережі рівня маршрутизації. Ці технічні реалізації відмічаються по декільком аспектам. Техніка маршрутизації визначає, коли і як внутрішні комутатори визначають множину входів та виходів та час, впродовж якого компоненти повідомлення можуть передаватись по цим маршрутам. Ці механізми поєднуються з механізмами управління потоком з метою синхронізації передачі одиниці інформації між маршрутизаторами і через маршрутизатори для передачі інформації через мережу. Управляючий потік тісно пов'язаний з алгоритмами управління буфером, який визначає як повідомлення буферизується при запитах та звільненнях і, як результат, визначають як повідомлення управляються при блокуванні в мережі.

2.2 Синтез безпосередньо зв'язаних топологічних мереж

Безпосередньо зв'язані топологічні мережі є ідеальним підходом з точки зору побудови масштабованих систем. Але ефективність таких систем значною мірою залежить від множини показників топологій відповідних систем. До таких показників в першу чергу будемо відносити кількість $\boldsymbol{N}$ вузлів топології, ступінь $\boldsymbol{S}$ топології, діаметр $\boldsymbol{D}$ топології, середній діаметр $\overline{\boldsymbol{D}}$ топології, топологічний трафік $\boldsymbol{Q}$, вартість $\boldsymbol{C}$.

Топологію будемо представляти у вигляді графу $\boldsymbol{G(X,Y)}$, де $X = \{0,1,\ldots,(N-1)\}$ – множина вершин, множина ребер $Y = \{0,1,\ldots,(E-1)\}$.

Діаметр (*D)* – це мінімальна відстань між найбільш віддаленими вершинами графу. Наприклад, в тривимірному гіперкубі найбільш віддаленими є вершини 0-7, 1-6, 2-5, 3-4 і для передачі даних від джерела до приймача потрібно 3 кроки, тобто діаметр $D = 3$.

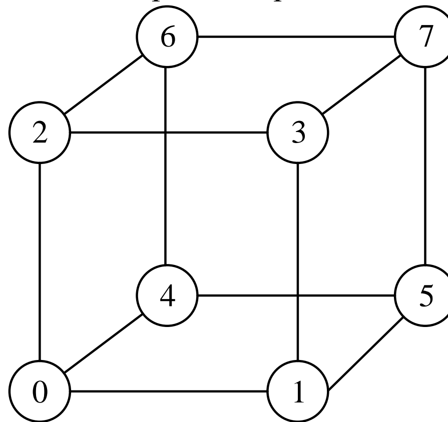


Рис. 2.5. Тривимірний граф.

Ступінь (*S)* – це максимальне число ребер, які інцидентні одній вершині. Наприклад в графі, що представлений на рис. 2.6 в якості таких вершин являються вершини 1 і 3, а отже ступінь $S = 3$

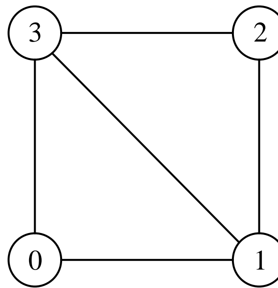


Рис 2.6. Граф зі ступенем S=3

Середній діаметр $(\overline{\boldsymbol{D}})$ - це деяка усереднена відстань між вершинами графа, яка в загальному вигляді може визначатися на основі наступної формули

$$\overline{\boldsymbol{D}} = \frac{\sum_{i=0}^{N} \sum_{j=i+1}^{(N+i-1) modN} d_{ij}}{N(N-1)} \qquad (2.1)$$

Для регулярного (однорідного) графу ця формула спрощується та для будь-якого i може бути представлена в наступному вигляді:

$$\overline{\boldsymbol{D}} = \frac{\sum_{j=i+1}^{(N+i-1) modN} d_{ij}}{N-1} \qquad (2.2)$$

В якості прикладу розглянемо обчислення середньої відстані для тривимірного гіперкубу наведеного на рис. 2.5. Оскільки граф однорідний, то середня відстань може бути визначена як середня відстань між будь-якою вершиною і та всіма іншими (N-1) – вершинами. Нехай і = 0, тоді середня відстань між і-тою вершиною визначається на основі кістякового (остовного) дерева, яке виходить із 0-ї вершини (рис. 2.7).

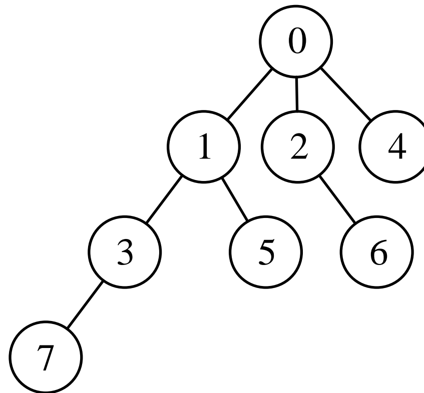


Рис. 2.7. Остовне дерево

Тоді у відповідності до (2.2)

$$\overline{D} = \frac{1 \cdot 3 + 2 \cdot 3 + 3 \cdot 1}{7} = \frac{12}{7}$$

У випадку неоднорідного графа цю процедуру слід виконати для кожної вершини, тобто N разів, скласти отримані результати та отриману суму розділити на N.

Топологічний трафік визначає потенційне середнє завантаження ребер повідомленнями (пакетами). Так, якщо припустити, що кожен вузол може в один і той самий момент часу передавати в мережу тільки один пакет, то одночасно в мережі може бути не більше N пакетів. Кожен пакет буде проходити середню відстань $\overline{\boldsymbol{D}}$. Отже, для передачі усіх пакетів без затримок, необхідно $\overline{D}N$ ребер. В дійсності, в графі є $R = \frac{N \cdot S}{2}$ ребер, тоді трафік

$$Q = \frac{\overline{D}N}{\frac{N \cdot S}{2}} = \frac{2\overline{D}}{S},$$

Для тривимірного гіперкубу

$$Q = \frac{2 \cdot 12}{7 \cdot 3} = \frac{24}{21} = \frac{8}{7}.$$

Зі збільшенням числа вузлів N топологічний трафік буде наближатися до 1. Таким чином, можна зробити висновок, що гіперкуб, з точки зору показника Q, близький до ідеального рішення.

Вартість (C) топологічної організації значною мірою залежить від її ступеня і дорівнює $C = N \cdot S$. Тому на показник ступеня накладаються істотні обмеження. Зазвичай вважається, що значення ступеню не повинні перевищувати значення 6, і зазвичай знаходяться в межах $S = 4 - 6$.

В свою чергу значення діаметру D, істотно впливає на призначену для користувача продуктивність системи, а отже підлягає оптимізації. Звідси випливає, що для пошуку оптимальних рішень топологічних організацій, необхідно виконати пошук рішень з мінімальними значеннями мультиплікативного крітерію $S \cdot D$, а при однакових значеннях показника $S \cdot D$ вибрати ті рішення, у яких топологічний трафік ближче до 1.

В даний час існує велика кількість топологічних організацій. На їх основі успішно вирішені завдання побудови мультипроцесорних і мультикомп'ютерних системи. Однак з підвищенням числа процесорних і комп'ютерних елементів, що має місце в даний час, ці топології неприйнятні через занадто великий ступень кожного вузла або неприпустимо великий діаметр топологічної організації системи. Тому в цьому розділі розглянемо різні підходи до синтезу нових топологічних організацій з мінімальними значеннями показника $S \cdot D$ і топологічних трафіків близьких до 1.

2.3 Синтез безпосередньо зв'язаних топологічних мереж

Існує безліч методів синтезу топологічних мереж. В даній роботі розглянуті найбільш вживані методи, до яких, зокрема, слід віднести методи синтезу на основі латинських квадратів, на основі кодових перетворень, на основі систем лінійних рівнянь, на основі інтеграції різних стандартних топологій, на основі ієрархічних побудов і деякі інші.

Найбільш простим з них є синтез топології на основі латинського квадрата, суть якого полягає в тому що деяка послідовність $(0,1, \ldots, (n-1))$ натуральних чисел так розподілена по матриці розміщення $(n \times n)$, що в рамках будь-якого рядка або будь-якого стовпця одні й ті ж числа не зустрічаються. Тоді топологія, яка побудована на основі латинського квадрата, буде визначатися вибраними стовпцями, тобто кожен елемент виділеного лівого стовпчика буде зв'язуватися з елементами вибраних стовпців відповідних рядків. Так, для прикладу, який представлено на рис. 1.13, відповідна топологія представлена на рис. 1.14.

Вибрані стовпці

0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0
2	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	1
3	3	4	5	6	7	8	9	10	11	12	13	14	15	0	1	2
4	4	5	6	7	8	9	10	11	12	13	14	15	0	1	2	3
5	5	6	7	8	9	10	11	12	13	14	15	0	1	2	3	4
6	6	7	8	9	10	11	12	13	14	15	0	1	2	3	4	5
7	7	8	9	10	11	12	13	14	15	0	1	2	3	4	5	6
8	8	9	10	11	12	13	14	15	0	1	2	3	4	5	6	7
9	9	10	11	12	13	14	15	0	1	2	3	4	5	6	7	8
10	10	11	12	13	14	15	0	1	2	3	4	5	6	7	8	9
11	11	12	13	14	15	0	1	2	3	4	5	6	7	8	9	10
12	12	13	14	15	0	1	2	3	4	5	6	7	8	9	10	11
13	13	14	15	0	1	2	3	4	5	6	7	8	9	10	11	12
14	14	15	0	1	2	3	4	5	6	7	8	9	10	11	12	13
15	15	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Рис. 1.13. Латинський квадрат

Ні по рядках, ні по стовпчиках немає повторення значень.

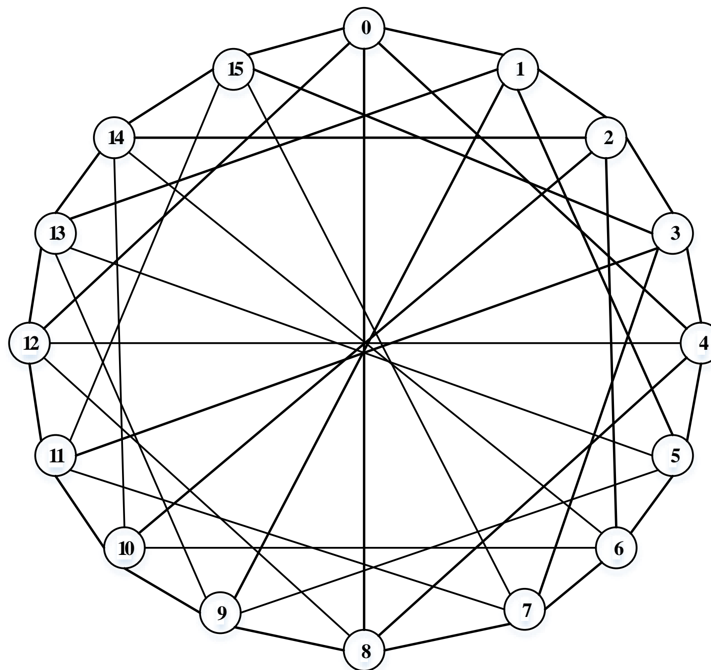
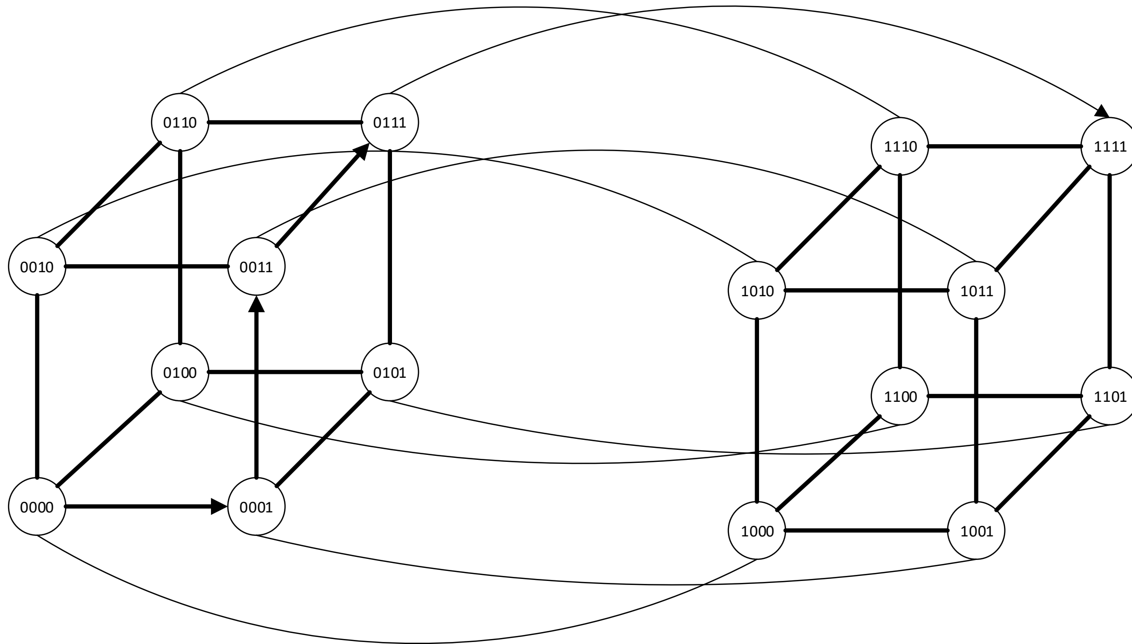


Рис. 1.14. Топологія, яка заснована на латинському квадраті.

Синтез на основі латинського квадрата, який представлений на рис. 1.13.

Алгоритм маршрутизації (наприклад, для гіперкуба):



0000 →1111
0000 →0001→0011→0111→1111

Якщо якісь вузли вийдуть з ладу, то робимо відкат і йдемо за наступною одиницею (більш старшою), після підсумовування поточного вузла з вузлом отримувачем по mod 2.

Синтез топології на основі кодових перетворень

Розглянемо наступне кодове перетворення

$$\begin{cases} a_0\ \ a_1\ a_2\ a_3 \\ a_1\ \ a_2\ a_3\ 0 \\ a_1\ \ a_2\ a_3\ 1 \\ 0\ \ a_0\ a_1\ a_2 \\ 1\ \ a_0\ a_1\ a_2 \end{cases}, \tag{1.17}$$

Яке визначає зсув 4 бітного коду спочатку вліво із запам’ятовуванням 0 або 1 на позиції, що звільнюються, потім вправо з аналогічним запам'ятовуванням позицій, що звільнюються.

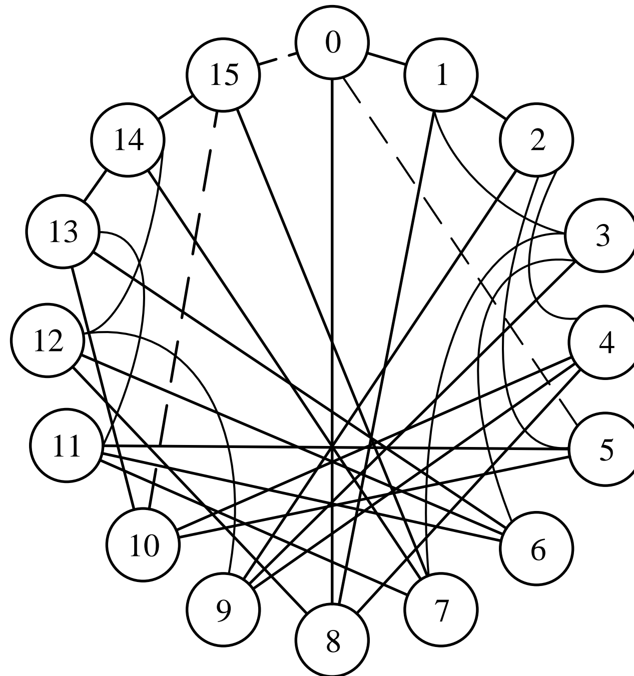


Рис. 1.15. Топологічна організація на основі кодових перетворень.

Наприклад,
Кодове перетворення для верш. 0:
<u>0000</u>
0000 (0)
0001 (1)
0000 (0)
1000 (8)

Таким чином, вузол «0000» в синтезуємій топології буде з'єднаний з вузлом «0001» і «1000». Тут виключається зв'язкам типу петля.

Аналогічно і інші вершини:

<u>0001</u> 0010 0011 0000 1000	<u>0010</u> 0100 0101 0001 1001	<u>0011</u> 0110 0111 0001 1001	<u>0100</u> 1000 1001 0010 1010	<u>0101</u> 1(?)010 1011 0010 1010	<u>0110</u> 1100 1101 0011 1011	<u>0111</u> 1110 1111 0011 1011	<u>1000</u> 0000 0001 0100 1100
<u>1001</u> 0010 0011 0100 1100	<u>1010</u> 0100 0101 0101 1101	<u>1011</u> 0110 0111 0101 1101	<u>1100</u> 1000 1001 0110 1110	<u>1101</u> 1010 1011 0110 1110	<u>1110</u> 1100 1101 0111 1111	<u>1111</u> 1110 1111 0111 1111	

Враховуючи, що формування топології здійснюється на основі зсувних перетворень з запам'ятовуванням позиції, що звільняються, нулями або одиницями, то це визначає найпростіший спосіб маршрутизації, який буде здійснюватися на основі послідовного заповнення коду джерела кодами приймача при зсуві кодів джерела або приймача вліво або вправо

$0010 \leftarrow 1100$ $\quad$ $1100 \rightarrow 0010$
$0101 \leftarrow 1000$ $\quad$ $0110 \rightarrow 0001$
$1011 \leftarrow 0000$ $\quad$ або $\quad$ $0011 \rightarrow 0000$
$0110 \leftarrow 0000$ $\quad$ $0001 \rightarrow 1000$
$1100 \leftarrow 0000$ $\quad$ $0000 \rightarrow 1100$

Нехай потрібно передати інформацію з вузла з номером 2 (0010) в вузол з номером 12 (1100), тоді в даній топології можна виділити два дерева, які доповнюють один одного.

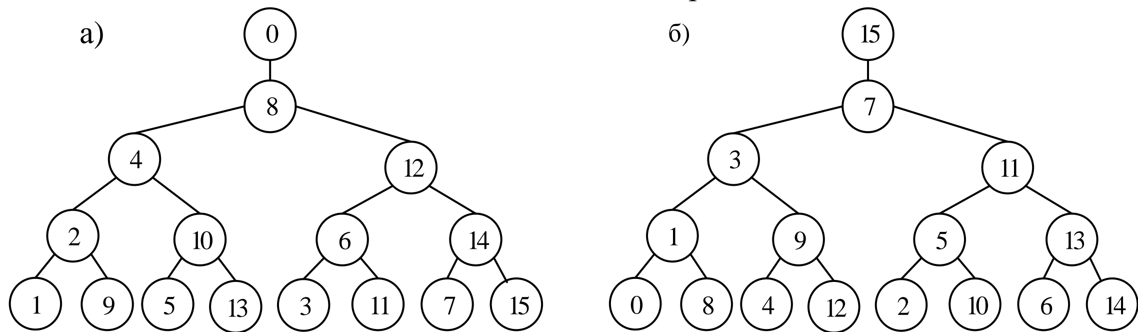


Рис. 1.16 Представлення топології, яка показана на рис. 1.15.

Що стосується маршрутизації, то при виході з ладу деякої вершини маршруту слід вибрати дерево в якому вона є кінцевою, підніматися по дереву вгору до вершини «0000» або «1111», а потім спускатися на вершину яка є кінцевою.

Тоді для маршрутизації з вузла 2 до вузла 12:
0010 ← 1100
0101 ← 1111
1011 ← 1110
0111 ← 1100

1100 ← 1111 піднялися на саму гору
0110 ← 0111
0011 ← 0011
0001 ← 1001
0000 ← 1100

Маршрут з 2 > 12 по першому дереву:
0010 ← 0000
0100 ← 0000
1000 ← 0000
0000 ← 0000

1100 → 0000 піднялися на саму гору
0110 → 0000
0011 → 0000
0001 → 1000
0000 → 1100

Особливість даних дерев полягає в тому що в дереві на рис. 1.16а кінцеві вершини є непарними, а в дереві рис. 1.16б парними. Це означає що переходячи від одного дерева до іншого ми завжди можемо зробити дефектну вершину кінцевою, тобто вихід будь-якої вершини з ладу призведе лише до виключення з системи тільки цієї вершини. Такі маршрути називають альтернативними маршрутами. Відмовостійкість можна збільшити за рахунок переходу від 2 до 4 системи.

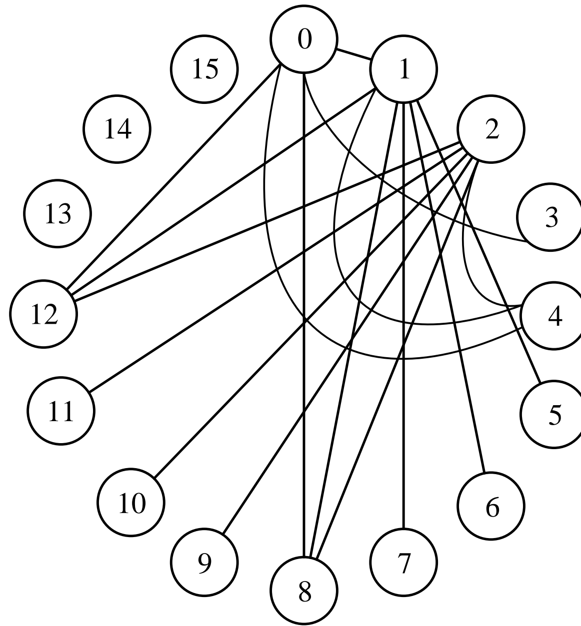


Рис. 1.17. Топологічна організація

Формуємо зв'язки для нульової вершини:

	$\underline{a_1 a_2}$
00	$a_2 0$
00	$a_2 1$
01	$a_2 2$
02	$a_2 3$
03	$0 a_1$
10	$1 a_1$
20	$2 a_1$
30	$3 a_1$

Аналогічно для вершин 01, 02:

01	02	
10	20	
11	21	
12	22	
13	23	. . . .
00	00	
10	10	
20	20	
30	30	

І так далі все йде за тим же принципом. Тепер у нас є 4 альтернативних дерева з 00, 11, 22, 33.

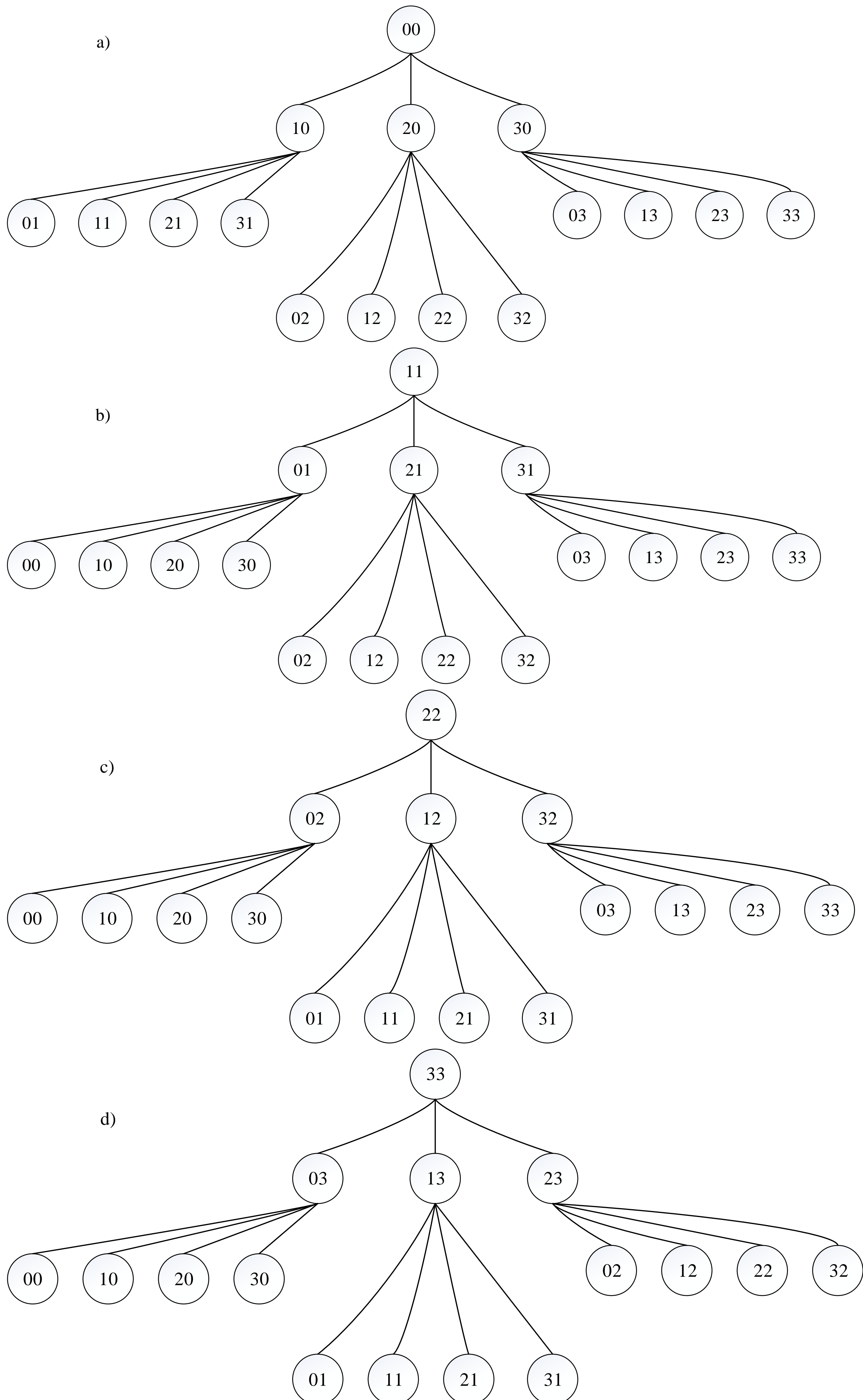
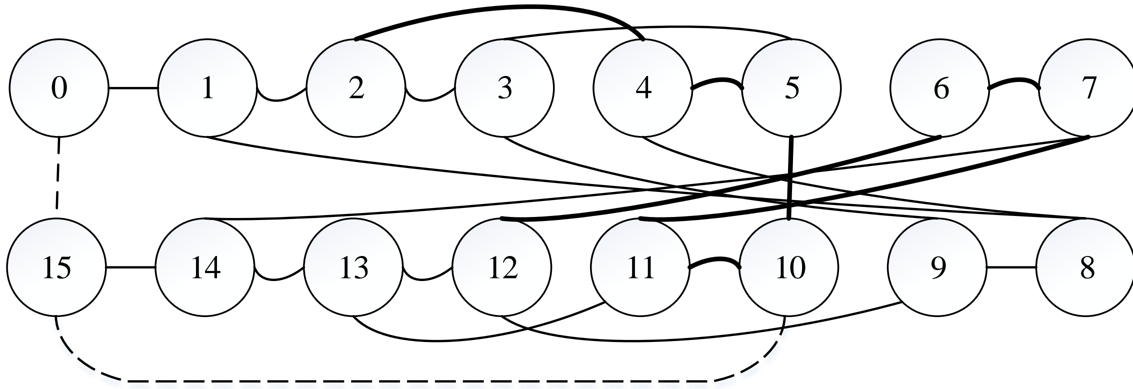


Рис. 1.18. Представлення топології, показаної на рисунку 1.17.

Наприклад, якщо вершина 20 (8 вершина) вийшла з ладу в першому дереві, в усіх інших деревах вершина 20 - кінцева вершина. Тут дуже проста маршрутизація, і система відмовостійка.

Розглянемо варіант з кластерами по два елементи. Нехай є множина кластерів по два елементи, один з яких парний, а другий - непарний (0-1, 2-3, 4-5,…). Елементи кластерів пов'язані між собою повною системою зв'язку, а решта пов'язані на основі циклічного зсуву.



Зв'язки на основі циклічного зсуву:

a_0 a_1 a_2 a_3

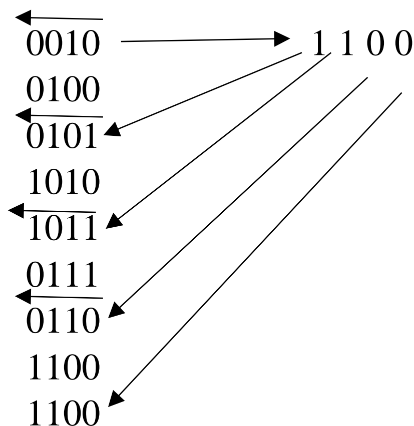
a_1 a_2 a_3 a_0

У нашому варіанті:

0000	-----	0001	0010	-----	0011
		0010	0100	-----	0110
0100	-----	0101	0110	-----	0111
1000		1010	1100		1110
1000	-----	1001	1010	-----	1011
0001		0011	0101		0111
1100	-----	1101	1110	-----	1111
1001		1011	1101		

Для симетрії додамо зв'язки $0 \leftrightarrow 15, 0 \leftrightarrow 5, 10 \leftrightarrow 15$.

Приклад маршрутизації $0010 \rightarrow 1100$:



Такий алгоритм маршрутизації має місце тільки у випадку, якщо дефектні вузли відсутні. При їх наявності алгоритм суттєво ускладняється.
Введемо наступні позначення:

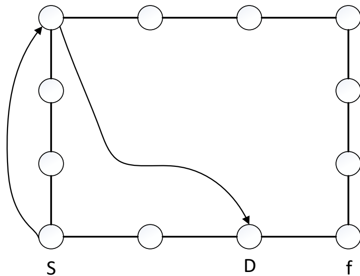
S f D

pt - шлях
S – джерело
D – приймач.
f – дефектний елемент
w(t) – вага, яка дорівнює кількості одиниць в коді вузла

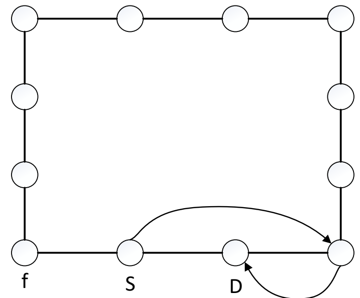
Так: w(0101) = 2
wt(1110) = 3

Правила:

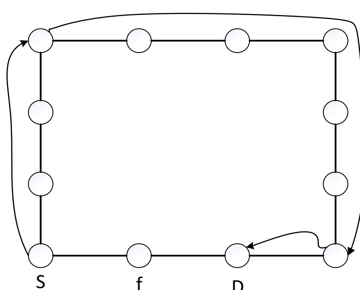
a) $wt(f) > wt(S),\ wt(f) > wt(D)$
$S \rightarrow pt(S,D) \rightarrow \emptyset \rightarrow pt(\emptyset,D) \rightarrow D$
Щоб була уніфікована система, то завжди передаємо з S в ∅, а з ∅ в D.



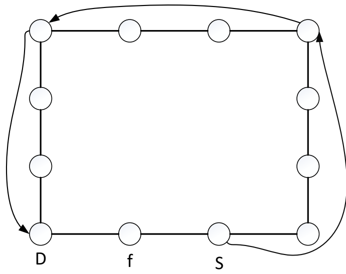
b) $wt(f) < wt(S)$ и $wt(f) < wt(D)$
$S \rightarrow pt(S,n-1) \rightarrow (n-1) \rightarrow pt(n-1,D) \rightarrow D$



c) $wt(f) > wt(S)$ и $wt(f) < wt(D)$
$S \rightarrow pt(S,0) \rightarrow \emptyset \rightarrow (n-1) \rightarrow pt(n-1,D) \rightarrow D$

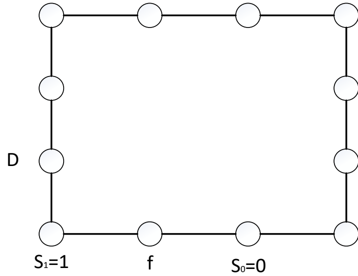


d) $wt(f) < wt(S)$ и $wt(f) > wt(D)$
$S \rightarrow pt(S,n-1) \rightarrow (n-1) \rightarrow \emptyset \rightarrow pt(0,D) \rightarrow D$



e) $wt(f) \ = \ wt(S)$ и $wt(f) \ > \ wt(D)$

if $\ f \cdot S_0 = \begin{cases} 1 & path\ as\ a \\ 0 & path\ as\ d \end{cases}$



e) $wt(f) \ > \ wt(S)$ и $wt(f) \ < \ wt(D)$

if $\ S_0 = \begin{cases} 1 & path\ as\ c \\ 0 & path\ as\ b \end{cases}$

g) $wt(f) = wt(D)$ и $wt(f) > wt(s)$

$$if\ D_0 = \begin{cases} 1 \to a \\ 0 \to c \end{cases}$$

h) $wt(f) = wt(D)$ и $wt(f) < wt(s)$

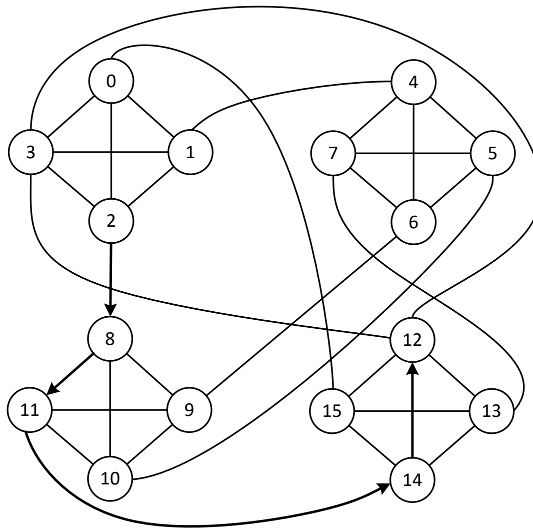
$$if\ D_0 = \begin{cases} 1 \to d \\ 0 \to b \end{cases}$$

i) $wt(f) = wt(S) = wt(D)$

$$if\ D_0 = \begin{cases} S_0 = 0, D_0 = 0 \to b \\ S_0 = 0, D_0 = 1 \to d \\ S_0 = 1, D_0 = 0 \to c \\ S_0 = 1, D_0 = 1 \to a \end{cases}$$

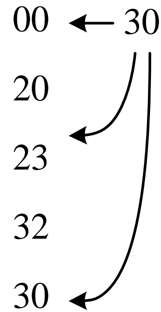
Ці 9 правил - при виході елементів з ладу.

Якщо кластери по 4 вузли:

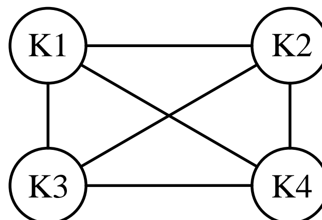


00	01	03	03	10	11	12	13
00(33)	10	20	30	01	11(22)	21	31
20	21	22	23	30	31	32	33
02	12	22	32	03	13	23	33(00)

Маршрутизація:



Якщо взяти кластери, то між ними теж повна система зв'язків:

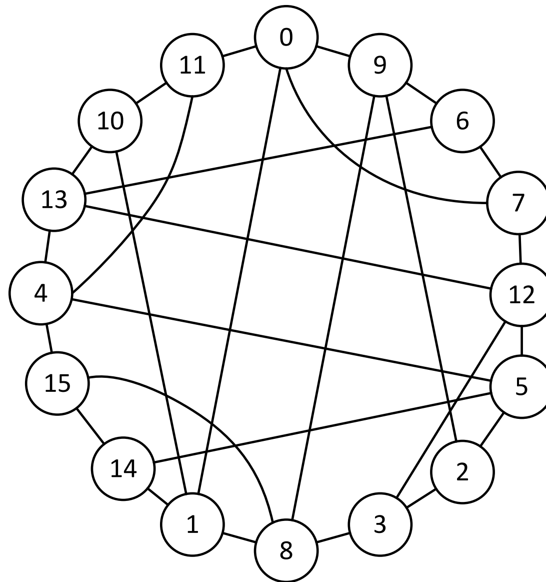


Відповідно, можна нарощувати рівні кластерів. Також це дає велику кількість альтернативних маршрутів.

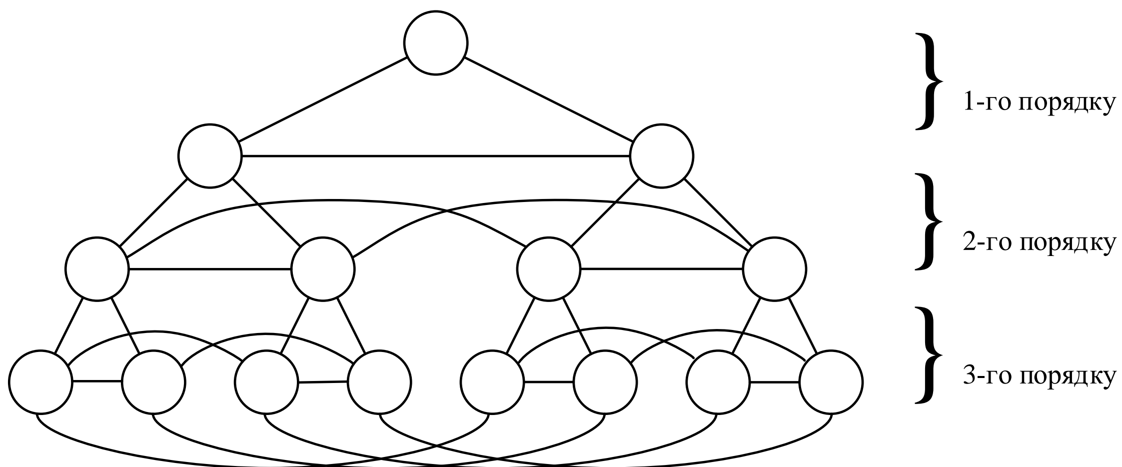
Синтез топології на основі лінійних рівнянь

$y = 5x + 9$

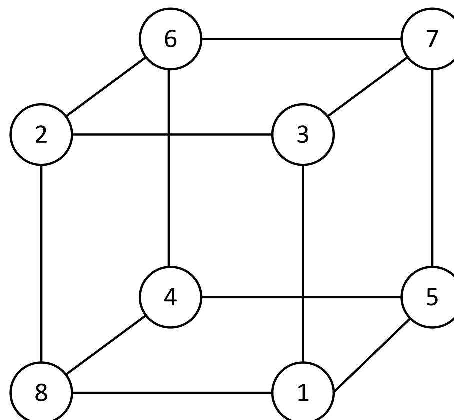
$y = 9x + 16$



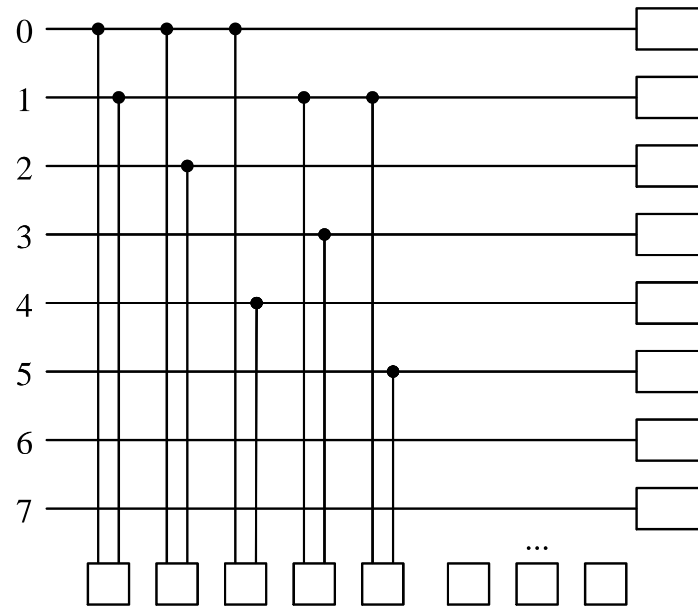
Як правило, всі ці технології не підходять для реальних ситуацій за своїми характеристиками. Наприклад, з точки зору управління найзручніше дерево. Його можна перетворити в гіперкуби відповідного порядку.



Як правило, ми розглядаємо вершинно-орієнтовані топології. Так, вершинно-орієнтований гиперкуб:



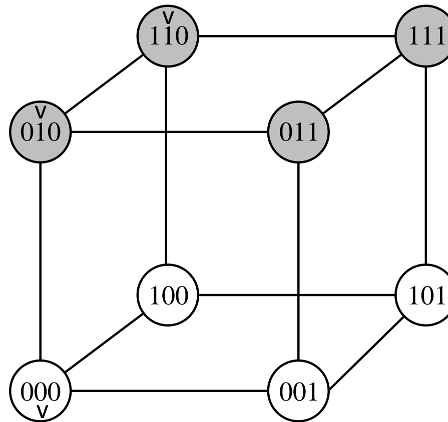
Реберно-орієнтований гиперкуб:


0
1
2
3
4
5
6
7
...

Функції маршрутизації даних.

Перестановка:

Функція перестановки (exchange): $E_k(b_m, \dots, b_k, \dots, b_1) = (b_m, \dots, b_k, \dots, b_1)$ для $1 \le k \le m$.



Тасування (shuffle):

4 варіанта.

Ідеальне тасування (perfect shuffle). Циклічний зсув вліво.

$$S(b_m, b_{m-1}, \dots, b_1) = (b_{m-1}, b_{m-2}, \dots, b_1, b_m) .$$

Відсутність тасування (unshuffle). Циклічний зсув вправо.

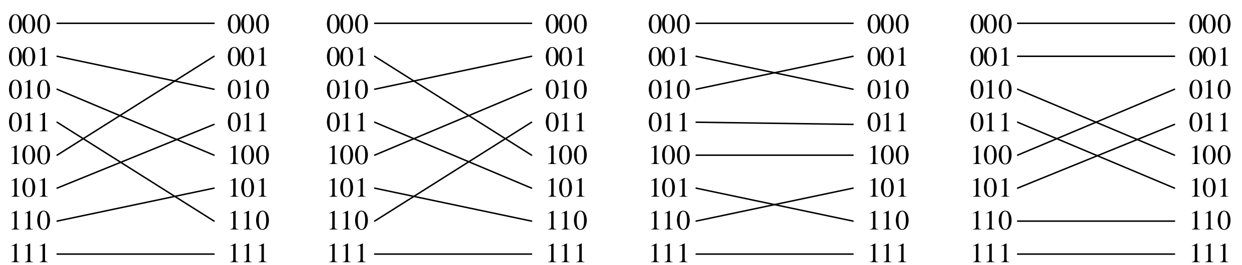
$$U(b_m, b_{m-1}, \dots, b_1) = (b_1, b_m, \dots, b_2)$$

Субтасування (subshuffle) по і-му входу.

$$S_i(b_m, b_{m-1}, \dots, b_i, \dots, b_1) = (b_m, \dots, b_{i+1}, b_{i-1}, \dots, b_i)$$

Супертасування (supershuffle) по і-му біту.

$$S^i(b_m, b_{m-1}, \dots, b_i, \dots, b_1) = (b_{m-1}, \dots, b_{m-i+1}, b_m, b_{m-i}, \dots, b_1)$$



Функції маршрутизації даних.

1. Перестановка (exchange): $E(b_m, b_{m-1}, \dots, b_k, \dots, b_1) = \left(b_m, b_{m-1}, \dots, \overline{b_k}, \dots, b_1\right)$

2. Тасування (shuffle):

а) ідеальне тасування (perfect shuffle). Зсув вліво.

$$S(b_m, b_{m-1}, \dots, b_1) = (b_{m-1}, b_{m-2}, \dots, b_1, b_m) .$$

б) відсутність тасування (unshuffle).

$$U(b_m, b_{m-1}, \dots, b_1) = (b_1, b_{m-1}, \dots, b_2)$$

в) субтасування по і-му біту (subshuffle).

$$S_i(b_m, b_{m-1}, \dots, b_i, \dots, b_1) = (b_m, b_{m-1}, \dots, b_{i+1}, b_1, b_{i-1}, \dots, b_i)$$

г) супертасування по і-му біту (supershuffle).

$$S^i(b_m, b_{m-1}, \dots, b_i, \dots, b_1) = (b_{m-1}, \dots, b_{m-i+1}, b_m, b_{m-i-1}, \dots, b_1)$$

3. Баттерфляй (butterfly):

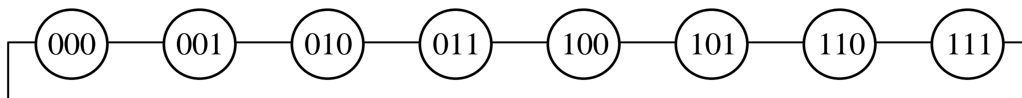
$$B(b_m, b_{m-1}, \dots, b_1) = (b_1, b_{m-1}, \dots, b_m) .$$

4. Реверсування бітів (bit reversal):

$$R(b_m, b_{m-1}, \dots, b_1) = (b_1, b_2, \dots, b_m) .$$

5. Зсув (shift):

$$SH(x) = (x+1) \ mod \ N$$



6. ILLIAC – IV

$$R_{+1} = (i+1) mod \ N$$

$$R_{-1} = (i-1) mod \ N$$

$$R_{+r} = (i+r) mod \ N \qquad r = \sqrt{N}$$

$$R_{-i} = (i-r) mod \ N$$

7. Циклічний зсув (barrel shift)

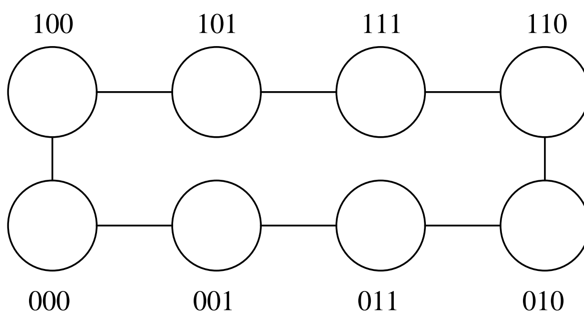
$$B_{+1}(j) = (j + 2^i) mod \ N, 0 \ \le j \ \le (N-1)$$

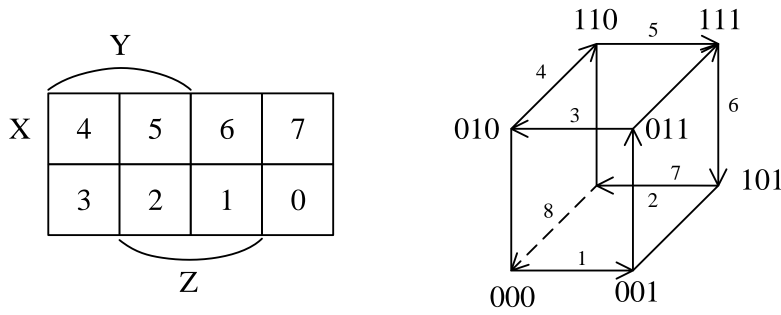
$$B_{-i}(j) = (j - 2^i) mod \ N, 0 \ \le i \ \le \log_2(N-1)$$

відсутня 1 сторінка (копія).

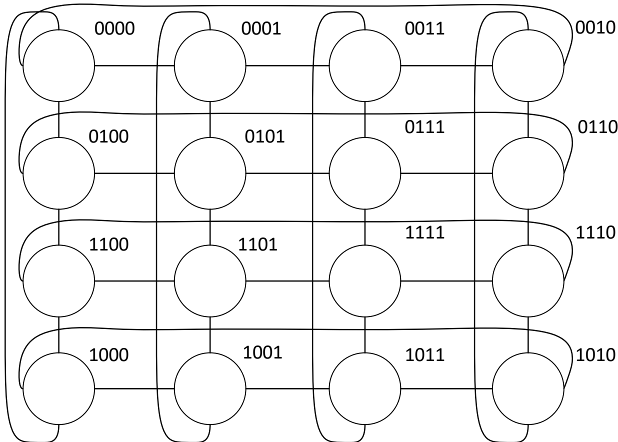
Відображення різних топологій в гіперкубі

Приклад 1: Відображення кільця в гіперкуб.

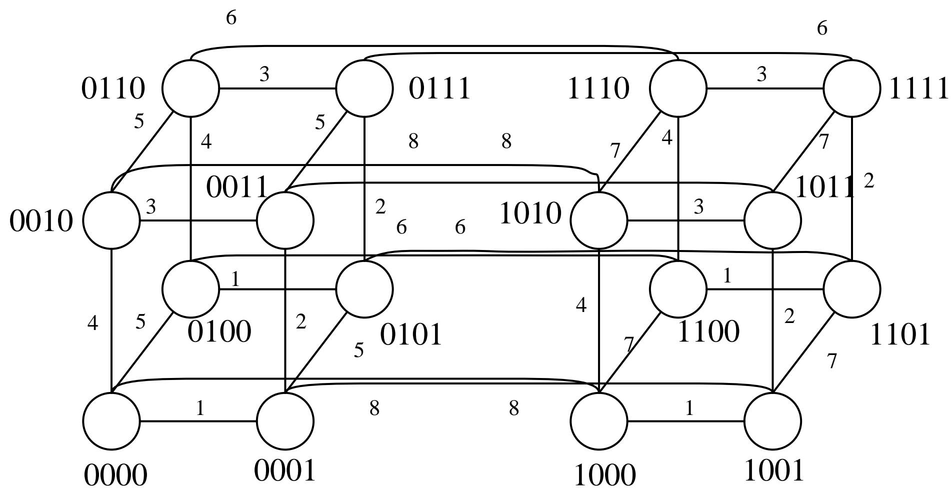




Приклад 2: Відображення Mesh в гіперкуб.



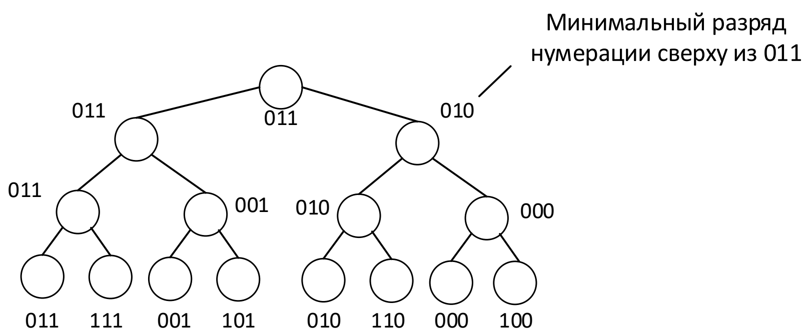
Представимо у вигляді гіперкубу.



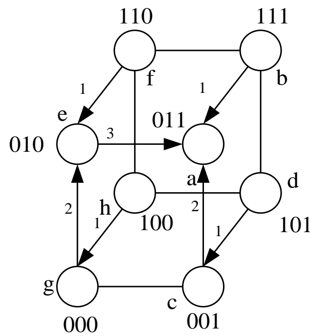
1. Відображаємо рядки.

2. Відображаємо стовпці.

Приклад 3: Відображаємо дерево на гіперкуб.



Відображення



Техніка комутації

Техніка комутації різниться за розмірами фізичних і логічних потоків управляючих одиниць. У загальному випадку кожне повідомлення може поділятися на пакети фіксованої довжини. Пакети, в свою чергу, можуть поділятися на управляючі елементи потоку даних або $flits$. Так як ширина каналу обмежена, то для передачі 1 flit, то може знадобитися безліч циклів (тактів) роботи каналу. $Phit$ - це кількість одиниць інформації, яка може передаватися через канал за один такт. Таким чином $flit$ - це логічна одиниця інформації, $phit$ - це фізична одиниця даних, тобто число бітів, які можуть передаватися протягом одного такту. Наприклад: повідомлення складається з 6 пакетів,6 $flits/packet$ і 2 $fits/flit$ (рис.)

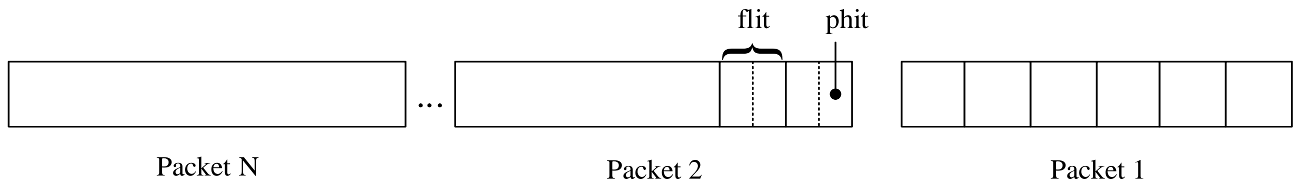


Рис. Повідомлення

Відносини між розмірами пакетів $flits$ і $phits$ різні в різних машинах. Так в IBM SP2 $flit$ дорівнює байту і розмір $flit$ і $phit$. В Cray T3D кожен $flit$ складається з 8-ми 16-бітових $phits$. Приклад: простий 4-фазовий асинхронний протокол пересилки з кодпідтвердженням.

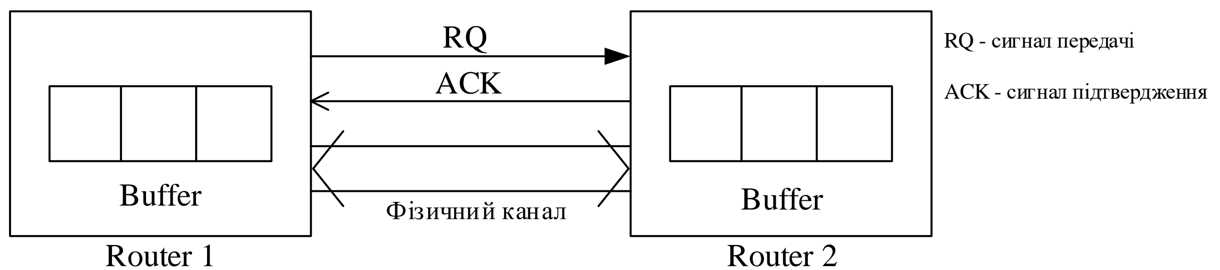


Рис. Протокол пересилки з кодпідтвердженням

Основи техніки комутації

З метою порівняння для кожного методу комутації будемо розглядати L-бітові повідомлення, розмір $flit$ дорівнює розміру $phit$, $flit = W_{bits}$, розмір заголовка дорівнює 1 $flit$. Таким чином, розмір повідомлення дорівнює $L + W$ бітів. Маршрутизатор вирішує питання маршрутизації на протязі t_r секунд. Частотні можливості каналу між двома

маршрутизаторами дорівнюють $B\ H2$, тобто пропускна здатність каналу одно BW біт в секунду, а затримка поширення через канал дорівнюють $t_w = \frac{1}{B}$. Вибір одного з маршрутів в маршрутизаторі, тобто внутрішня затримка маршрутизатора або затримка комутації дорівнює t_S. Таким чином, протягом t_S, W-бітовий $flit$ може бути переданий з входу маршрутизатора на його вихід. Припускається, що відстань між джерелом і приймачем дорівнює D лінкам.

Комутація каналів ефективна в тому випадку, коли повідомлення передаються рідко і є довгими, тобто в тому випадку коли час передачі є суттєво більшим у порівнянні з часом установки з'єднання. Недолік комутації каналів полягає в тому, що фізичні шляхи займаються на весь період передачі повідомлення і можуть при цьому блокувати інші повідомлення.

Комутація каналів.

Комутація каналів вимагає резервування шляху від джерела до приймача до початку передачі даних. Це досягається на основі передачі заголовка по мережі. Цей маршрутизаторний зонд включає в себе адресу призначення і деяку управляючу інформацію. Він просувається до вузла призначення, резервуючи при цьому фізичні канали. Коли зонд досягає вузла призначення, то при цьому буде встановлено повний маршрут просування інформації, після чого підтвердження зонда будуть передані приймачу. Даний шлях може бути звільнений або вузлом призначення, або останніми бітами повідомлення. Текст після встановлення зв'язку може передаватися з повним використанням пропускної спроможності. Підтвердження може передаватися або по тій же фізичній лінії, що і повідомлення, або може бути передбачені спеціальні лінії підтвердження.

Даний вид комутації може бути ефективним, в разі якщо повідомлення передаються рідко і є довгими. Недолік - можливість блокування адреси передачі. Тут маршрутний зонд буферизується в кожному маршрутизаторі, а повідомлення - ні, так як по суті для передачі повідомлення надається єдиний провід.

Основна затримка при передачі повідомлення для випадку комутації каналів визначається часом формування (установки) шляху і послідовністю інтервалів часу, необхідних на передачу даних. Якщо маршрутна інформація буферизується в кожному маршрутизаторі, то дані не буферизуються. Не потрібно буферизування даних, так як канал діє як єдиний дротовий зв'язок між джерелом і приймачем. Цей зв'язок може використовуватися як в асинхронному, так і синхронному режимі. У цьому випадку передача одного $flit$ від джерела до приймача визначається тактовою частотою в разі синхронного режиму або швидкістю сигналізуючих сигналів при асинхронному режимі з підтвердженням. Період сигналізуючих сигналів або синхронізуючих сигналів повинен бути більше часу поширення сигналів через ланцюг. Це накладає практичні обмеження на швидкість методу комутації каналів як функції від розмірів системи.

$$t_{kk} = t_{setup} + t_{data}$$

$$t_{setup} = D(t_r + 2(t_S + t_W))$$

$$t_{data} = \frac{1}{B}\left\lceil\frac{L}{W}\right\rceil$$

Коммутация каналов.

Коммутация каналов требует резервирования пути от источника до приемника до начала передачи данных. Это достигается на основе передачи заголовка по сети. Этот маршрутизаторный зонд включает в себя адрес назначения и некоторую управляющую информацию. Он продвигается до узла назначения, резервируя при этом физические каналы. Когда зонд достигает узла назначения, то при этом будет установлен полный маршрут продвижения информации, после чего подтверждения зонда будут переданы приемнику.

Дальний путь может быть освобожден либо узлом назначения, либо последними битами сообщения. Текст после установления связи может передаваться с полным использованием полосы пропускания. Подтвержение может передаваться, либо по той же физической линии, что и сообщение, либо может быть предусмотрены специальные линии подтверждления.

Данный вид коммутации может быть эффективным в случае когда сообщение передаются редко и являются длинными. Недостаток – возможность блокирования адреса передачи. Здесь маршрутный зонд буферируется в каждом маршрутизаторе, а сообщения – нет, так как по сути для передачи сообщения представляется единый провод.

Основная задержка при передапчи сообщения для случая комутации каналов определяется временем формирования (установки) пути и последовательно интервалов времени, необходимых на передачу данных. Если маршрутная информация буферируется в каждом маршрутизаторе, то данные не буферируютися. Не требуется буферирование данных, так как канал действует как единая проводная связь между источником и приемником. Эта связь может использоваться как в асинхронном, так и синхронном режиме. В этом случае передача одного флита от источника к приемнику определяется тактовой частотой в случае синхронного режима или скоростью сигнализирующих сигналов при асинхронном режиме с подтверждением. Период сигнализирующих сигналов или синхронизирующих сигналов должен быть больше времени распространение сигналов через цепь. Эта накладывает практические ограничения на скорость метода комутации каналов как функции от размеров системы.

Коммутация пакетов.

Если при коммутации каналов после установки соединения сообщение передается в полном объеме, то при коммутации пакетов сообщение разделяется и передается пакетами фиксированной длины, например 128 байт. Первые несколько байтов пакета создают маршрутную и управляющую информацию и рассматриваются как заголовки пакета. Каждый пакет индивидуально передается от источника к приемнику. Пакеты полностью буферизуются в каждом промежуточном узле перед тем, как будут направлены к следующему узлу. Поэтому данный метод передачи часто называют $store-and-forward\ (SAF)$ коммутацией. В заголовке имеет место информация о следующем маршрутизаторе, которая используется для определения выходного линка, через который он будет проходить.

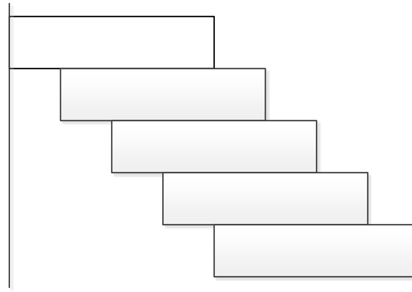
Данный метод коммутации эффективен в случае, если сообщения являются короткими и следуют с большой частотой. В отличии от коммутации каналов, в данном случае блокирования передачи других сообщений не будет. Много пакетов, принадлежащих сообщению могут быть в сети одновременно, даже если первый пакет еще не пребыл к месту назначения. Однако разделение сообщений на пакеты приводит к некоторым накладным расходам. Так, каждый пакет должен маршрутизироваться в каждом промежуточном узле. Для упорядочивания пакетов в приемнике нужна информация о соответствующей последовательности пакетов в сообщении.

Виртуальная конвейерная коммутация (Cut-through (VCT)).

Коммутация пакетами основана на предположении, что пакеты должны приниматься в целом прежде, чем результаты маршрутизации могут создать и направить пакет к приемнику. Это не всегда так. Рассмотрим 128-байтовый пакет. В отсутствии физического канала на 128 байт, передача пакета по физическому каналу потребует множество циклов. Однако, первые несколько байтов будут содержать информацию для маршрутизации, которая обычно доступна через несколько циклов. Поэтому маршрутизатор может начинать передавать заголовок и следующие за ним байты данных сразу после получения данных для маршрутизации и в случае свободного выходного буфера.

В действительности соединения не должны даже буферизоваться на выходе, а могут конвейерным способом передаваться на вход следующего маршрутизатора до получения полного пакета в текущем маршрутизаторе. Такой метод коммутации называется VCT – коммутацией.

$$t_{VCT} = D\ (t_r + t_s + t_w) + \max(t_s, t_w) \left\lceil \frac{L}{W} \right\rceil$$



Wormhole Switching

Необходимость буферизации целый пакетов внутри маршрутизатора может создавать трудности для конфигурирования малых, компактных и быстрых маршрутизаторов. При червячной коммутации пакетов, пакеты сообщения также перемещаются через сеть в конвейерные решения. Однако, буферизация внутри router требует существенно меньших расходов, чем для VCT. Пакет сообщения разделяется на $flit$. $Flit$ – это единица управления потоком, а входные и выходные буферы в роутере достаточно велики, чтобы хранить несколько $flit$. Сообщение протекает через сеть на уровне флитов и обычно является достаточно большими, чтобы буферизоваться внутри роутера. Таким образом, в любой момент времени блокированное сообщение занимает буферы в нескольких роутерах. Основное различие между червячной коммутацией пакетов и VCT в том, что единицей управления потоком сообщения здесь является единственный флит, а следовательно, размер буферов является очень небольшим.

В отсутствии блокирования пакет сообщений передается по сети конвейерным способом. Рис. 2.12 иллюстрирует моментальный снимок соединения, передаваемого через роутеры R1, R2 и R3. Вх. и вых. буферы имеют глубину в 2 флита, а заголовок является 2-х флитовым. В роутере R3 сообщение A требует вых.канал, который испльзуется соединением B. Следовательно, сообщение A блокируется. Небольшие размеры буферов в каждом узле предполагают необходимость использования множества роутеров и тем самым блокирует другие сообщения.

$$T_{wormhole} = D(t_r + t_s + t_w) + max\ (t_s, t_w) \left\lceil \frac{L}{W} \right\rceil$$

Коммутация на принципе сумасшедшего почтальона.

VCT повышает производительность коммутации пакетов путем конвейеризации потока сообщений, оставляя при этом способность буферирования (буферизации?) целых пакетов сообщения. Wormhole обеспечивает дальнейшее понижение задержек на основе использования малых буферов VCT так, чтобы маршрутизация могла быть полностью управляемая внутри одиночного роутера, а следовательно, обеспечивала низкую задержку распространения, необходимую для плотно связанных параллельных процессов. Развитие этого направления связано с увеличением уровня конвейеризации на основе реализации механизма сумасшедшего почтальона для реализации минимально возможных задержек на одни узел.

Этот механизм наиболее понятен в контексте бит-последовательных каналов. Рассмотрим 2-D mesh сеть с пакетами, которые имеют $2\text{-}flit$ заголовки. Маршрутизация первоначально полностью выполняется вдоль измерения 0, а затем 1. Первоначальный флит заголовка содержит адрес узла назначения в направление "0". Когда сообщение достигнет этого узла, сообщение перенаправляется в измерение "1". Второй флит заголовка содержит адрес узла назначения по измерению "1". В VCT и wormhole флиты не могут передаваться до тех пор, пока все флиты заголовка не будут получены в текущем роутере. Механизмы сумасшедшего почтальона позволяет понизить задержку на одном узле путем конвейеризации на битовом уровне. Каждый бит заголовка также буферизуется локально.

Виртуальные каналы

Предыдущие механизмы коммуникации были описаны, в предположении, что сообщения или фрагменты сообщения были буферизированы на входе и выходе каждого физического канала. Эти буферы организованы как FIFO-очереди. Следовательно, если одно сообщение занимает буфер канала, то ни одно другое сообщение не имеет доступа к физическому каналу, даже если сообщение блокировано. Однако, существует другой вариант организации, при которой один физический канал может поддерживать несколько логических или виртуальных каналов. При этом каждый однонаправленный виртуальный канал реализуется на основе независимо управляемой пары буферов сообщений (рис.). Этот рисунок иллюстрирует два однонаправленных виртуальных канала в каждом направлении на основе одного физического канала. Рассмотрим $wormhole\ switching$ с сообщением в каждом виртуальном канале. Каждое сообщение может разделять физический канал на основе $flit - by - flit$. Протокол физического канала должен быть способным различать виртуальные каналы в рамках одного физического канала. Логически, каждый виртуальный канал действует так, как если бы соответствующий физический канал действовал в половину скорости. Виртуальные каналы были введены первоначально, чтобы решать проблему $dead\ lock$ в $wormhole\ switching\ networks$.

Виртуальные каналы могут также использоваться для уменьшения задержек сообщений и увеличения сетевой пропускной способности. На основе разделения физического канала сообщения могут распространяться быстрее за счет исключения (уменьшения) блокировок.

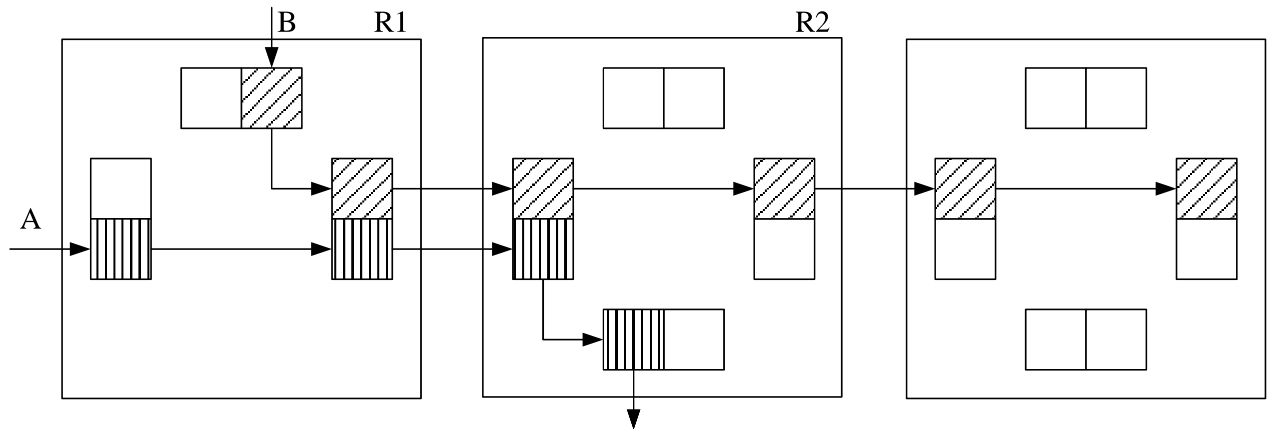


Рис. Пример

На рисунке показано два сообщения, пересекающие физический канал между маршрутизаторами R1 и R2. При отсутствии виртуальных каналов сообщения A будет блокировать сообщение B до тех пор, пока передача сообщения A не завершиться. Однако в данном случае два одно-$flit$-вых виртуальных канала мультиплексируются в рамках одного физического канала. При этом оба сообщения передаются без блокировок. Однако скорость распространения номинально будет в два раза меньше, чем при отсутствии разделения. В действительности использование виртуальных каналов подобно реализации множества программ, разделяющих CPU (режим разделения времени).

Можно выделить два случая, когда такое разделение приносит наибольший эффект. Рассмотрим случай, когда сообщение A временно заблокировано при протекании к текущему узлу. Тогда протокол физического уровня позволяет сообщению B использовать полную пропускную способность физического канала между маршрутизаторами. Кроме того, рассмотрим случай, когда сообщение A намного длиннее сообщения B. При этом сначала сообщение B будет разделять пропускную способность физического канала, затем A будет в полной мере использовать пропускную способность канала.

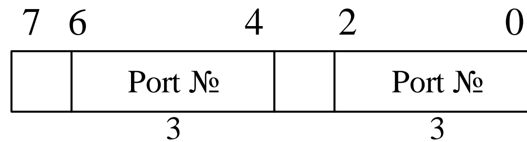
Подход, описанный в предыдущем параграфе, не имеет ограничений на использование виртуальных каналов. Использовавшиеся таким образом буферы будем называть виртуальными тропинками.

Гибридные механизмы коммуникации.

1. Буферизуемый wormhole switching.

Данная техника коммуникации объединяем особенности червячной коммутации и коммутации пакетов. Механизм коммутации и формат сообщений должны выбираться с учетом особенностей коммуникационных сетей, применяемых в системах с единственным? процессором. Эти сети являются многоканальными, и основаны на omega – сетях, использующих коммутацию 4×4 с дуплексными линками. Система на 16 процессоров образует блок, реализованный на двухканальном коммутаторе с 8 коммутирующими коммуникационными элементами. Этот модуль будем называть фреймом.

Пакеты сообщения могут быть переменной длины до 256 флитов. Каждый флит имеет длину в 1 байт и равен ширине физического канала. Первый флит сообщения содержит длину сообщения, а следующий флит содержит маршрутную информацию. Маршрутизация решается в источнике (sourse-based), где каждый содержит адрес выходного порта в промежуточных коммутациях. Для каждого фрейма имеется 1 $flit$ маршрутизации, т.е. для каждой группы из 16 процессоров.



Здесь двунаправленные линки, а следовательно каждый входной порт одновременно является и выходным. Биты 4-6 определяют выходной порт первого коммутатора, биты 0-2 – выходной порт второго коммутатора. Бит 7 используется для определения поля, который будет использоваться. Каждый фрейм требует 1 флит маршрутизации. Большие системы строятся на основе группирования фреймов.

Так как сообщений проходят через коммутаторы, то соответствующие флиты маршрутизации отбрасываются, сокращая сообщение. До тех пор, пока маршруты свободны от конфликтов, сообщения распространяются как в wormhole switching в междукоммутационном потоке при управлении на уровне флитов. Когда выходные каналы доступны, поток данных через коммутатор имеет ширину пути в 1 байт проходит внутренний коммутатор и далее передается на выходной порт. Когда сообщение блокируется, управление внутри коммутатора организовано на уровне 8-флитовых единиц, называется chunks (порции). Когда сообщение блокируется, создаются порции во входном порте коммутатора, которые затем через 64-битовый канал (кеш) передаются в локальную память. Далее в последовательном порядке, буферируемые порции передаются к выходному порту, где они преобразуются в последовательность флитов для передачи через физический канал.

Когда имеет место конфликт на уровне маршрутного узла, флиты блокируются внутри коммутатора в виде порций. Эти порции буферизуются динамически, распределяясь в центральной памяти. Память организована в виде списка линков для каждого выхода, где каждый элемент списка это сообщение, которое будет передаваться на выходной порт. Т.к. только первая порция сообщения содержит маршрутную информацию, каждое сообщение организовано как список порций. Таким образом, порядок флитов внутри пакета сообщений защищен.

Конвейерная коммутация каналов.

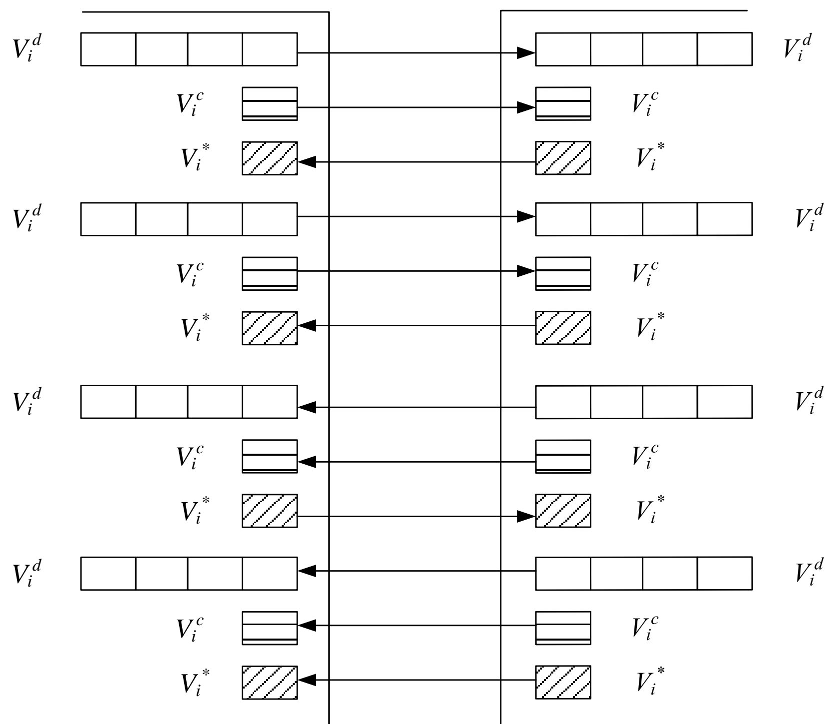
Во многих системах большее внимание уделяется не уменьшению задержек распространения и повышения пропускной способности сети, а вопросам отказоустойчивости сетевых компонентов (маршрутизаторов и линко). В случае wormhole- коммутации, заголовок, содержащий маршрутную информацию, устанавливает маршрут через сеть от источника до приемника. Флиты данных конвейеризуюся вдоль маршрутов, непосредственно следующих за заголовочным флитом. Если заголовок не может распространяться в связи с вышедшими из строя компонентами, то сообщение блокируется, занимая буферные ресурсы и блокируя другие сообщения. В тоже время адаптивная маршрутизация может облегчить данную проблему, не решая ее полном объеме. Это побуждает к разработке различных механизмов коммуникации.

Конвейерная коммутация каналов (PCS) комбинирует в себе коммутацию каналов и коммутацию wormhole. PCS отличается от коммутации каналов в том, что маршруты формируются на основе виртуальных каналов, а не физических. В PCS флиты данных не следуют непосредственно за заголовочным флитом, как это имеет место в wormhole switching. Следовательно, увеличение гибкости достигается на основе маршрутизации эталонного флита. Например, если имеет место блокирование в случае выхода из строя промежуточного маршрутизатора, то заголовок может быть возвращен в предшествующий маршрутизатор, и найти альтернативный маршрут, и освободить ранее задействованный канал.

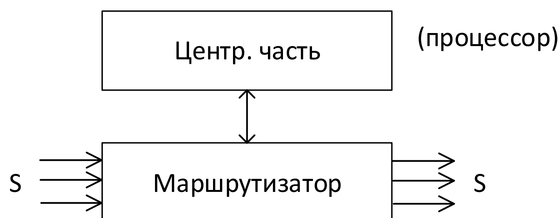
Новый выходной канал может теперь сделать попытку с помощью предшествующего маршрутизатора найти альтернативный путь к приемнику. Когда заголовок наконец достигает узла назначения, подтверждающий флит будет передан назад к источнику. Теперь флиты данных могут быть конвейерным образом передаваться так, как это имеет место в wormhole-коммутации. Этот подход является гибким в том плане, что заголовок может выполнят откат с целью исследования сети, резервируя и освобождая виртуальные каналы с целью установления маршрута к источнику, свободного от отказов.

Многомаршрутность путей в соединении с гибкостью PCS обеспечивают низкую задержку распространения с отказоустойчивостью.

В PCS мы будем различать управляющие флиты, т.е. флиты заголовка, подтверждающие флиты и флиты данных. Это различие поддержано моделью виртуальных каналов, в которых различаются флиты, управляющие трафиком и трафик флитов данных. Однонаправленные виртуальные каналы и составляют (объединяют) канал данных, соответствующий канал и комплементарный канал (V_i^d, V_i^c, V_i^*) и относятся к виртуальному трио-каналу. Маршрутный заголовок будет представлен как V_i^c, а флиты данных как V_i^d. Комплементарный канал V_i^* определяет подтверждающие флиты и откатные флиты.



Алгоритмы маршрутизации



Для фиксированных связей:

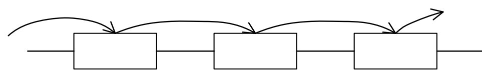
Маршрутизация – это прокладывание маршрута от источника к приемнику.

Будем выделать 3 этапа выполнения маршрутизации:

1) Инициализация (t_i)
   С инициализацией будем связывать выполнение таких операций:
   - формирование кадра
   - выполнение алгоритма маршрутизации
   - установление связи с маршрутизатором
   Для систем с фиксированной системой связи маршрутизатор выполняет только передачу. А в коммутируемых системах связей – вопросы маршрутизации.
2) Передача заголовка в соседний узел
3) Передача пакета в соседний узел

1) Маршрутизация на уровне сообщение (store-and-forward, SF)

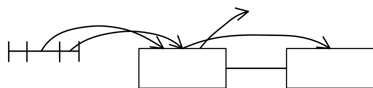
Имеется линейка узлов. Сообщение передается в один узнал полностью, а затем полностью передается в другой узел.



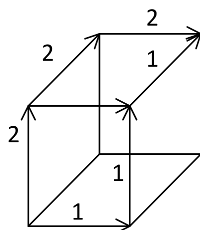
Последовательная передача

2) Конвейерный подход на уровне сообщение (cut-through)

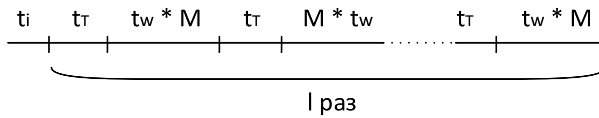
Так же, как и первый подход, но сообщение разделяется на части и передается по частям.



3) Параллельная маршрутизация



I. Последовательная маршрутизация



t_w * M – передача M слов длины t_w

t_i – время инициализации

t_T –

l – количество передач между узлами

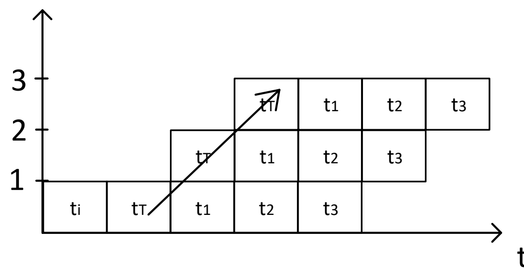
Общее время SF маршрутизации:

$t_{SF} = t_i + t_T * l + t_w * M * l$

Так как t_T очень мало, то

$t_{SF} \approx t_i + t_w * M * l$

II. Конвейерная маршрутизация



$t_{CT} = t_i + t_T * l + t_w * M$

$\Delta = t_w * M * l - t_\tau * M = t_\tau * M * (l-1)$ - разница между CT и SF

Вывод: Чем дольше l, тем более логично применять конвейерную маршрутизацию (CT).

Алгоритм один к одному (one-to-one)

$t_{SF}^{C} = t_i + t_w * M * \frac{N}{2}$ – Circle, N – количество узлов

$t_{SF}^{M} = 2t_i + 2t_w * M * [\frac{\sqrt{N}}{2}]$ – Mesh

$t_{SF}^{HC} = t_i + t_w * M * \log_2 N$ – Hyper Cube

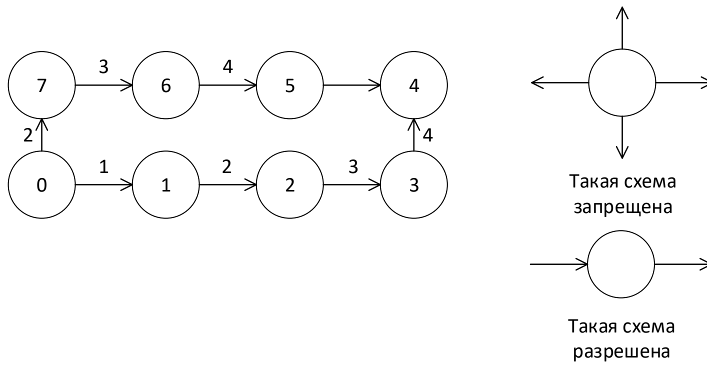
$t_{CT}^{C} = t_i + t_w * M + t_T * \frac{N}{2}$ – Circle

$t_{CT}^{M} = 2t_i + 2t_w * M + 2t_T * [\frac{\sqrt{N}}{2}]$ – Mesh

$t_{CT}^{HC} = t_i + t_w * M * \log_2 N$ – Hyper Cube

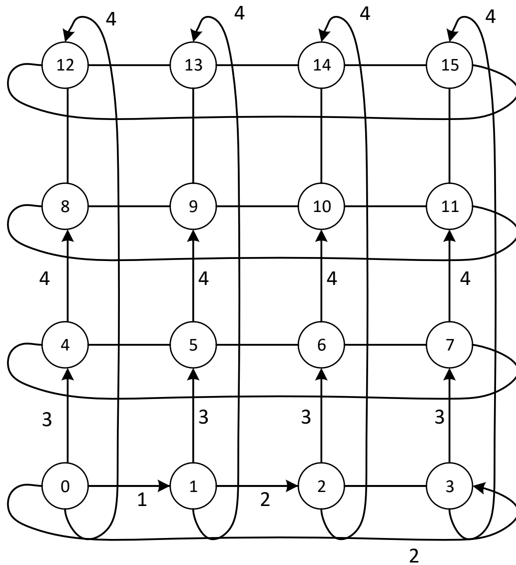
Маршрутизация один ко всем (one-to-all)

Рассмотрим кольцо



$$t_{SF}^{C} = (t_i + t_w * M) * \frac{N}{2}$$

Mesh-топология:

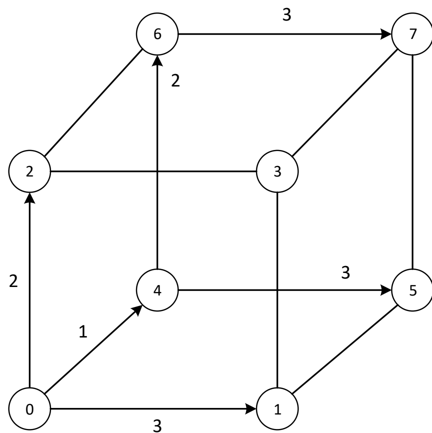


1,2 шаги – доставка сообщения во все столбцы

3,4 шаги – сообщение доставляется во все узлы

$$t_{SF}^{M} = 2(t_i + t_w M)\left\lceil\frac{\sqrt{N}}{2}\right\rceil$$

Гиперкуб (рассмотрим далее подробно):



$$t_{SF}^{HC} = (t_i + t_w M)\log_2 N$$

1. procedure ONE_TO_ALL_HC(d, my_id, M)
2. begin
3. mask: 2^d-1
4. for i:=d-1 downto ∅ do
5. begin
6. mask:=mask XOR 2^i
7. if (my_id AND mask) = ∅ then
8. if (my_id AND 2^i) = ∅ then
9. begin
10. msg_dest:=my_id XOR 2^i
11. send M to msg_dst
12. endif
13. else
14. begin
15. msg_source:=my_id XOR 2^i
16. recieve M from msg_source
17. endelse
18. endfor
19. end procedure ONE-TO-ALL-HC

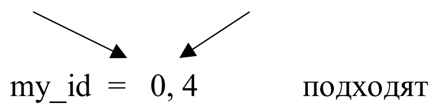
mask = 111; //(7_{10}) d – степень гиперкуба

i=2;

mask = $\begin{array}{c}111\\ \underline{111}\\ 011\end{array}$

Для всех узлов подстав. my-id:

$\begin{array}{c}000\\ \underline{011}\\ 000\end{array}$ $\begin{array}{c}001\\ \underline{011}\\ 001\end{array}$ … $\begin{array}{c}100\\ \underline{011}\\ 000\end{array}$ …



my_id = 0, 4 подходят

$\begin{array}{c}000\\ \underline{100}\\ 000\end{array}$ $\begin{array}{c}100\\ \underline{100}\\ 100\end{array}$

0 – подходит

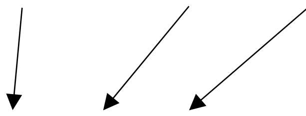
msg_dest = $\begin{array}{c}000\\ \underline{100}\\ 100\end{array}$ = 4

msg_source = $\begin{array}{c}100\\ \underline{100}\\ 000\end{array}$

i = 1:

mask = $\begin{array}{c}011\\ \underline{010}\\ 001\end{array}$

$\begin{array}{c}000\\ \underline{001}\\ 0\end{array}$, $\begin{array}{c}001\\ \underline{001}\\ 1\end{array}$, $\begin{array}{c}010\\ \underline{001}\\ 0\end{array}$ … $\begin{array}{c}110\\ \underline{001}\\ 0\end{array}$ …



0, 2, 6

У алгоритма есть недостаток – все начинаем с нуля.

Можно любую вершину преобразовать в a:

```
1.   procedure one_to_all(d, my_id, source, M)
2.     begin
3.           my_virtual_id = my_id XOR source;
4.           mask:=2^d-1;
5.           for i=d-1 downto ∅ do
6.           begin
7.                mask:= mask XOR 2^i
8.                if (my_virtual_id AND mask) = ∅ then
9.                    if(my_virtual_id AND 2^i) = ∅ then
10.                      begin
11.                          msg_dest = my_virtual_id XOR 2^i;
12.                          send M to (msg_dst XOR source);
13.                   endif
14.               else
                  …
```

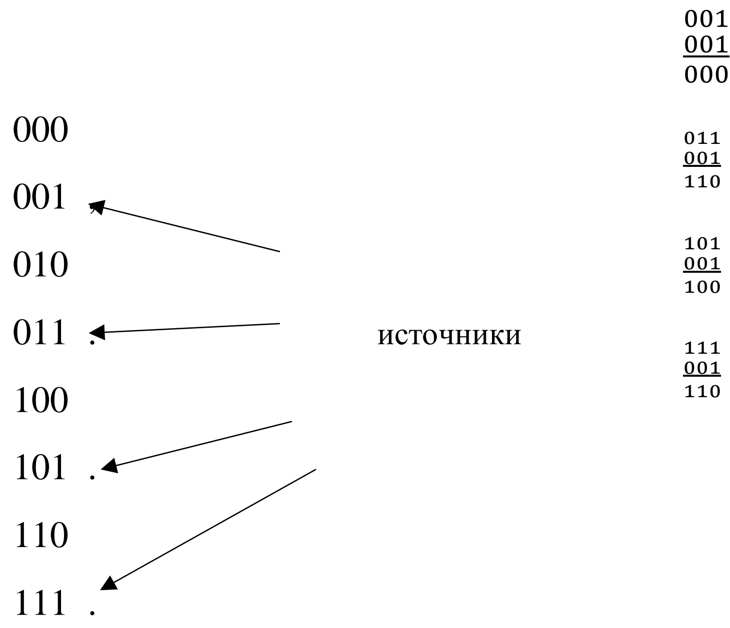
Пример опер. слож.

```
1.   procedure SINGLE_NODE_ACC_HC(d, m, my_id, M, sum)
2.   begin
3.       for j:= ∅ to m-1 do sum[j]:= M[j]
4.            mask:= ∅
5.             for i:=0 to d-1 do
6.                begin
7.                      if(my_id AND mask) = ∅ then
8.                           if (my_id AND 2^i) = ∅ then
9.                              begin
10.                                 msg_dest:=my_id XOR 2^i
11.                                 send sum to msg_dest
12.                            endif
13.                          else
14.                            begin
15.                               msg_source:=my_id XOR 2^i
16.                               recieve M from msg_source
```

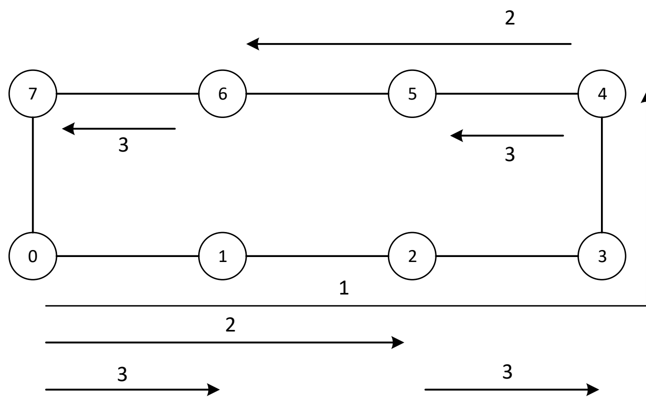
```
17.                              for j:= ∅ to m-1 do
18.                                   sum[j]+M[j]
19.                          endelse
20.                       mask:= mask XOR 2^i
21.               endfor
22.           endfor
23. end
```

Выполнение (Debug):

1) i=0



CT ONE_TO_ALL



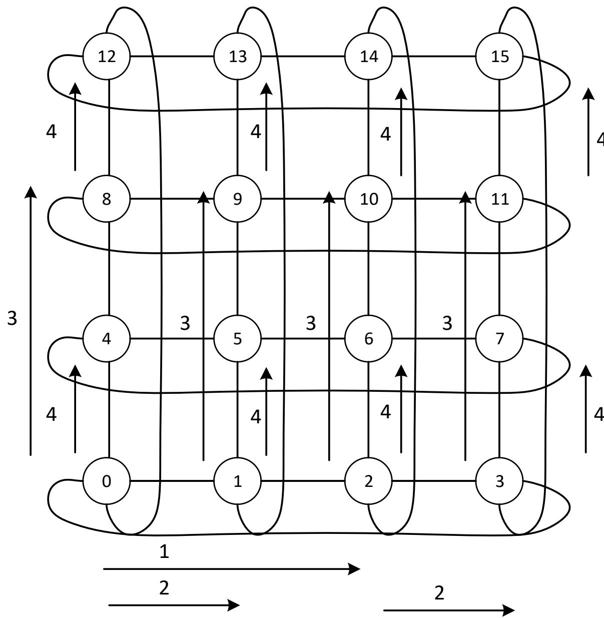
1	2	3	
$\frac{N}{2}$	$\frac{N}{4}$	$\frac{N}{8}$	L

$$t_{\text{ст}}^{\text{с}} = \sum_{i=1}^{\log N} (t_i + t_w M + t_T * \frac{N}{2^i}) = (t_i + t_w M) \log_2 N + t_T \frac{N(N-1)}{N} =$$

$$= (t_i + t_w M) \log_2 N + \text{t}_2 (N-1).$$

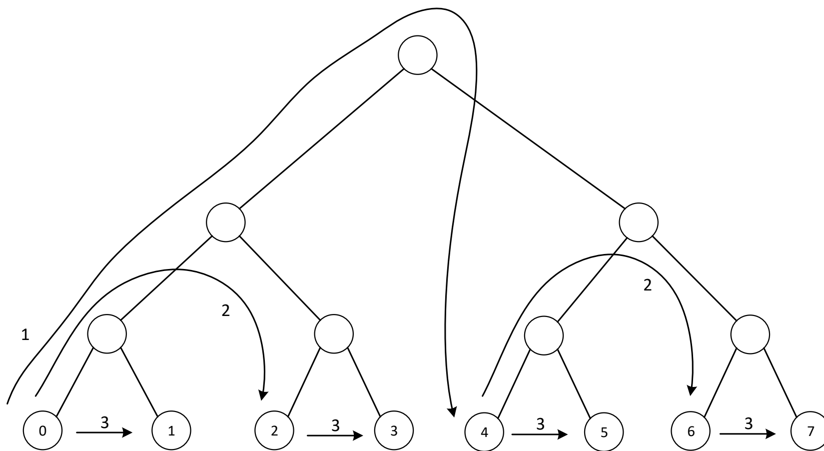
Следующая mesh-топология:

Делаем на основе строк и столбцов



$$t_{\text{ст}}^{\text{М}} = 2(t_i + t_w M)\log_2 \sqrt{N} + \mathrm{t}_T\left(\sqrt{N} - 1\right) = (t_i + t_w M)\log_2 N + \mathrm{t}_T\left(\sqrt{N} - 1\right)$$

Для дерева:

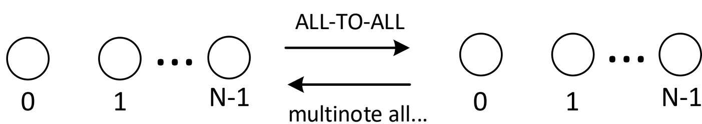


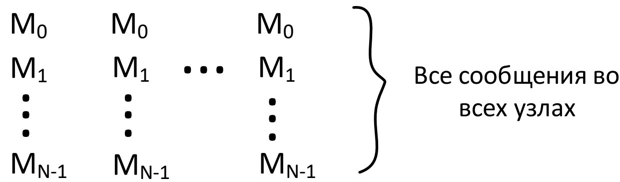
$$t_{\text{ст}}^{tree} = (t_i + t_w M + \mathrm{t}_T(\log_i N + 1))\log_2 N$$

Это для СТ-маршрутизации.

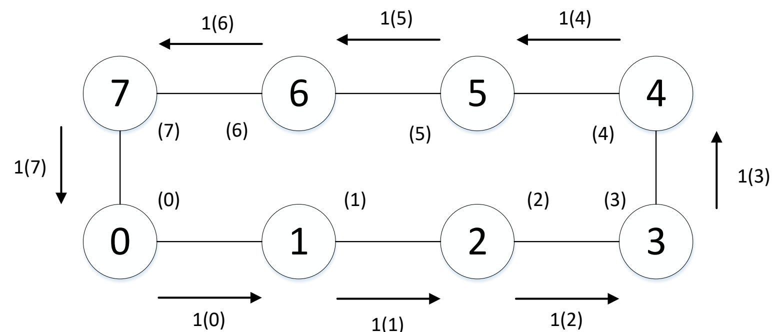
Лекция №4

ALL_TO_ALL

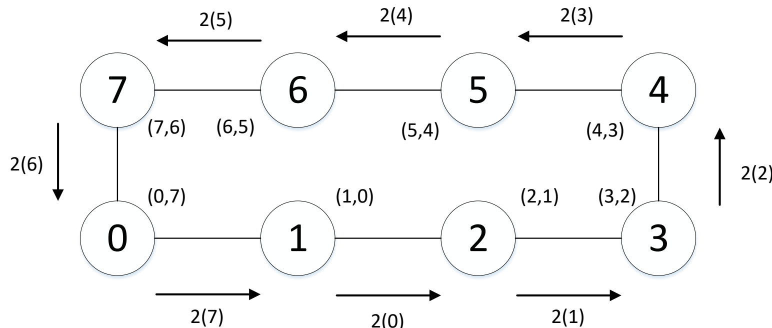




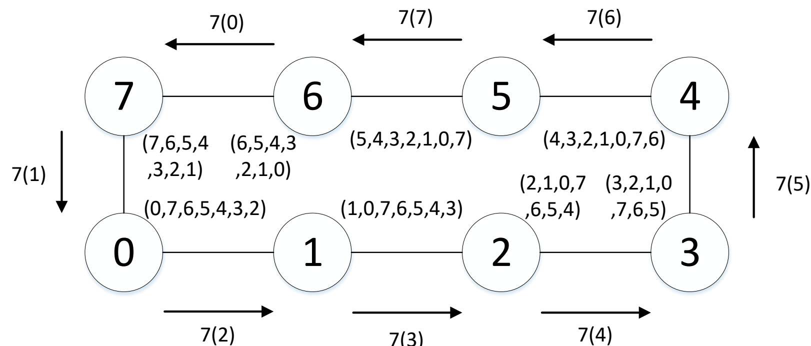

```
1.   Procedure ALL_TO_ALL_RING (my_id, my_msg, N, result)
2.   begin
3.           left := (my_id - 1) mod N;
4.           right := (my_id + 1) mod N;
5.           result := my_msg;
6.           msg := result;
7.           for i = 0 to N-1 do
8.                   begin
9.                           send msg to right;
10.                          send msg from right;
11.                          result := result U msg;
12.          end for;
13. end procedure ALL_TO_ALL_RING.
```



Первая пересылка



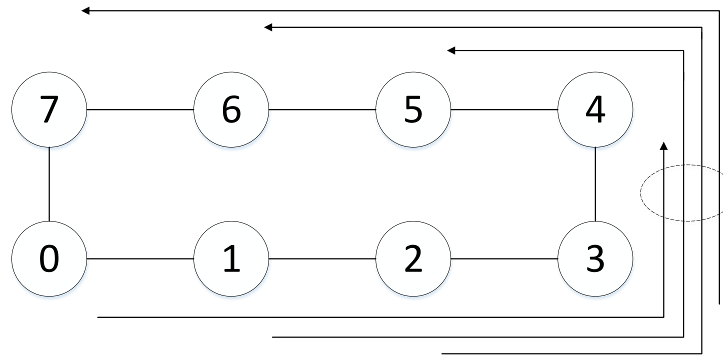
Вторая пересылка



Последняя седьмая пересылка

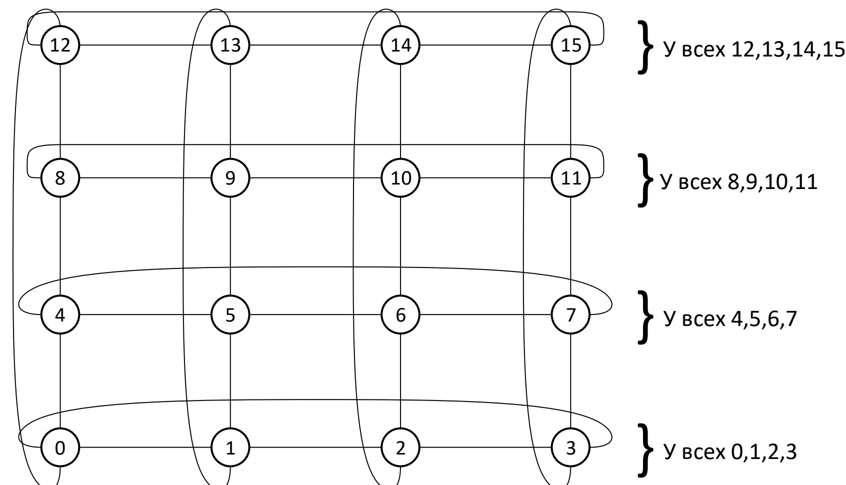
$$t_{SF}^{o} = (t_i + t_w * m) * (N - 1)$$

ALL_TO_ALL_CT (с конвейером)



Получается, что через одно ребро (3-4) необходимо передавать одновременно информацию. Соответственно в этом месте образуется затор. То есть для ALL_TO_ALL лучше всего использовать SF (store and forward)
Mesh-топология

Сначала рассылаем в рамках строк, как в кольце, получаем:



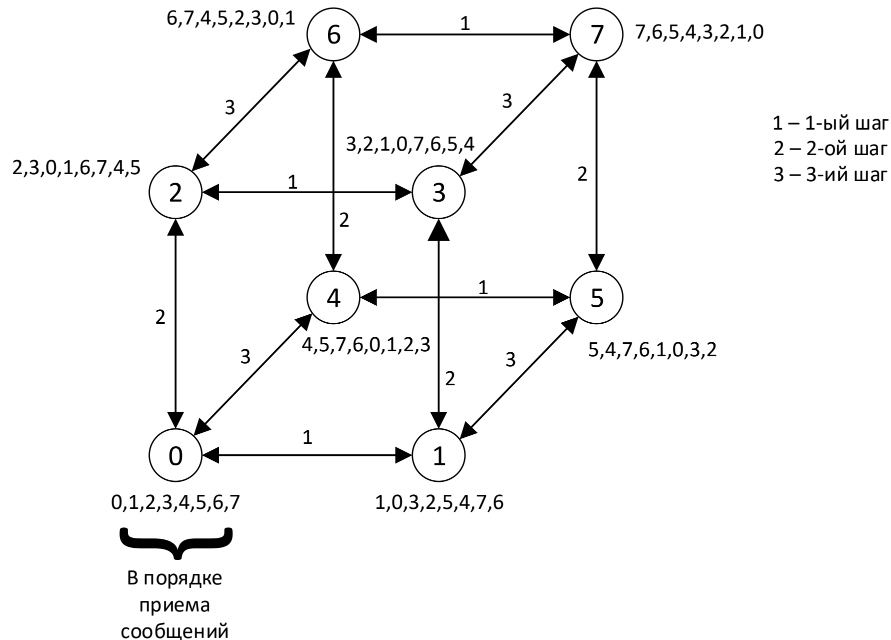
$$t_{SF}^{M} = (t_i + t_w * m) * \left(\sqrt{N} - 1\right) + \left(t_i + t_w * m * \sqrt{N}\right) * \left(\sqrt{N} - 1\right) =$$
$$= \left(2 * t_i + t_w * m * \left(\sqrt{N} + 1\right)\right) * \left(\sqrt{N} - 1\right) =$$
$$= 2 * t_i * \left(\sqrt{N} - 1\right) + t_w * m * (N - 1)$$

m – количество слов в сообщении

Гиперкуб:

```
1.  procedure ALL_TO_ALL_HC (my_id, my_msg, d, result)
2.  begin
3.          result := my_msg;
4.          msg := result;
5.          for i = 0 to d-1 do
6.                  begin
7.                          partner := my_id xor 2;
8.                          send msg to partner;
9.                          receive msg from partner;
```

10. result := result U msg;
11. end for;
12. end procedure ALL_TO_ALL_HC.



$$t_{SF}^{HC} = \sum_{i=1}^{d} \left(t_i + t_w * m * 2^{i-1}\right) = t_i * d + t_w * m * (N-1)$$

Пример:

Нахождение префиксной суммы

$n_0, n_1, \ldots, n_{N-1}$ – число в каждом узле (узел 0 – узел N-1 соответственно).

Префиксная сумма:

$$S_k = \sum_{i=0}^{k} n_i \,, k = 0, 1, 2, \ldots, N-1$$

Процедуру изменяем таким образом:

1. procedure PREFIX_SUM (my_id, my_msg, d, result)

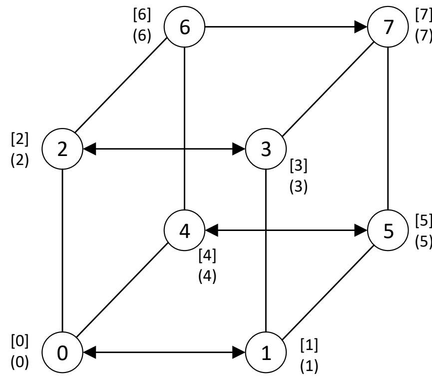
.

.

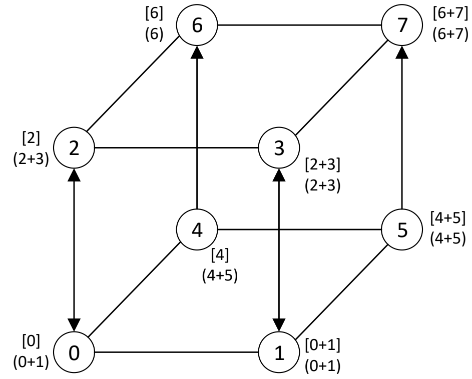
.

11. if (p < toe < my_id) then
12. result := result + number;

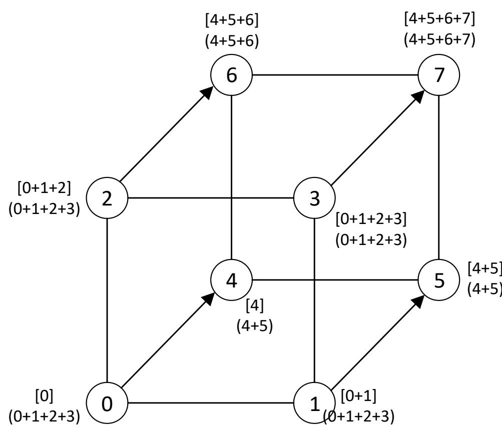
Лекция №5 (Продолжение)



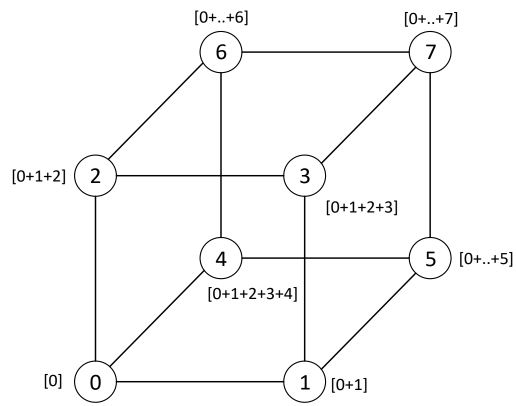
Изначальное распределение



Первая пересылка



Следующая пересылка



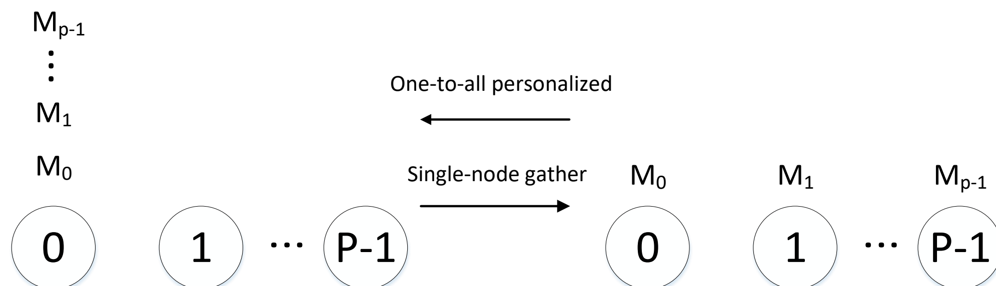
Последняя пересылка

То есть префиксную сумму можно делать только на основе пересылки.

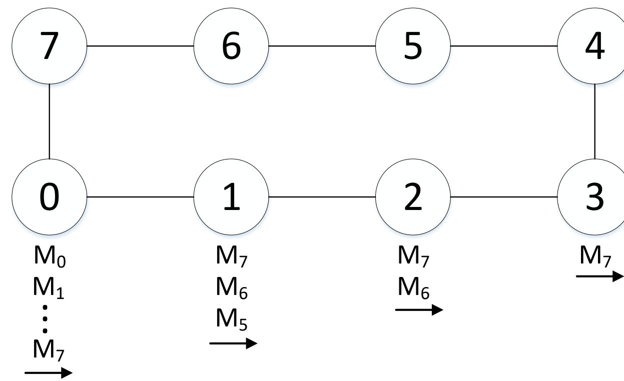
Есть и другое множество операций, которые можно делать на основе пересылки.

ONE_TO_ALL_PERSONALIZED

(Один всем с персональным назначением)



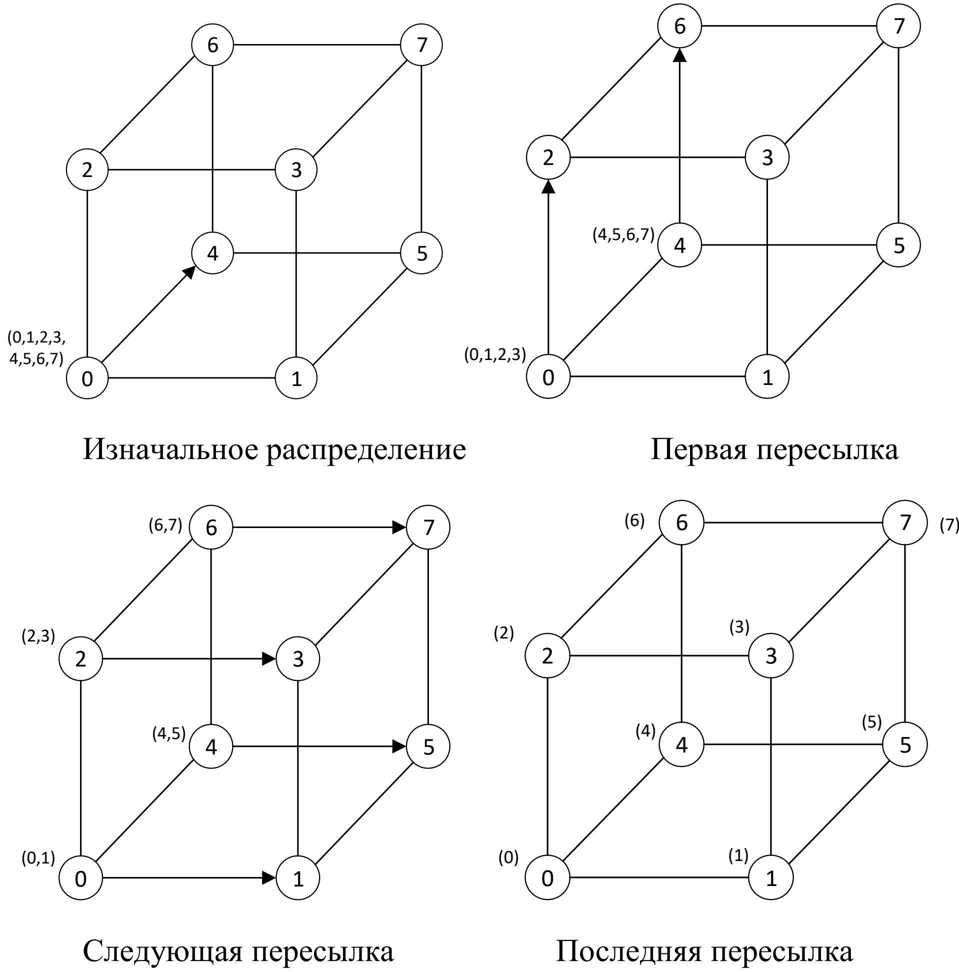
Все временные соотношение как в ALL_TO_ALL



Здесь конвейеризация не уместна

Для этого потребуется N-1 шагов, но тогда: $t = (t_i + t_w * m) * (N - 1)$

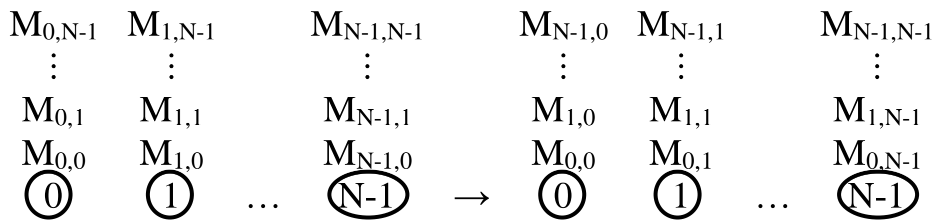
На основе гиперкуба:



Изначальное распределение   Первая пересылка

Следующая пересылка   Последняя пересылка

ALL_TO_ALL_PERSONALIZED

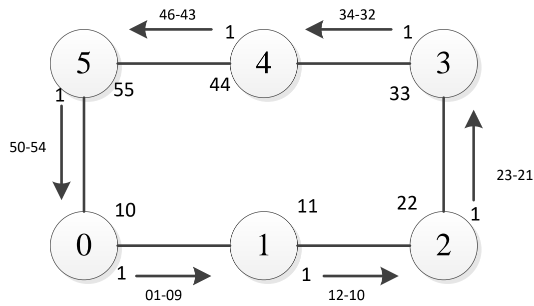


all_to_all_pers

По сути на основании опер. перес. получили транспонированные матрицы.

Кольцо:

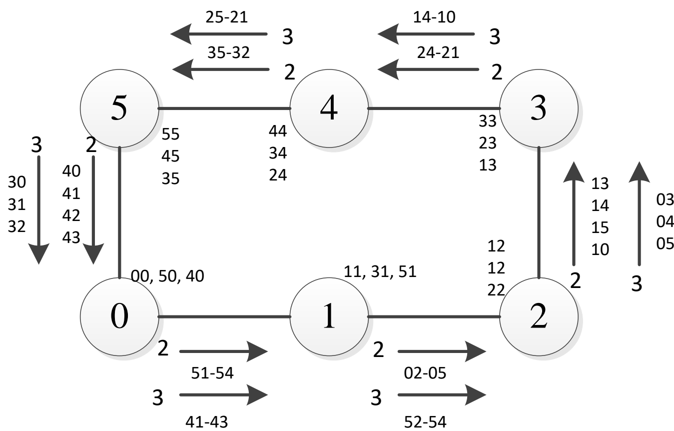
Первый шаг:



05	15	25	35	45	55
04	14	24	34	44	54
03	13	23	33	43	53
02	12	22	32	42	52
01	11	21	31	41	51
00	10	20	30	40	50

Выделенные элементы уже на месте!

Второй шаг; Третий шаг:

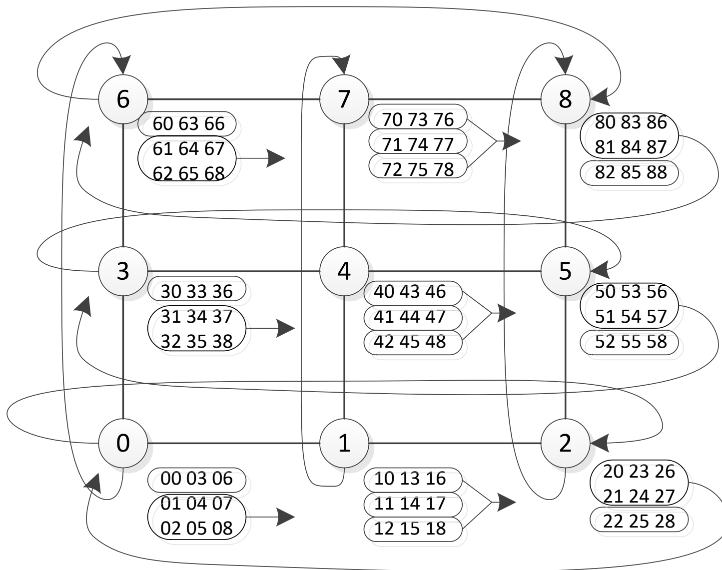


Четвертый т. д.

$$t = \sum_{i=1}^{N-1}\left(t_i + t_w m(N-i)\right) = = t_i(N-1) + \sum_{i=1}^{N-1} t_w m = t_i(N-1) + \frac{N}{2}(N-1)t_w m;$$

$$t_{SF}^{C} = (N-1)\left(t_i + \frac{N}{2}t_N m\right);$$

Mesh – топология:



Потом аналогично по столбцам.

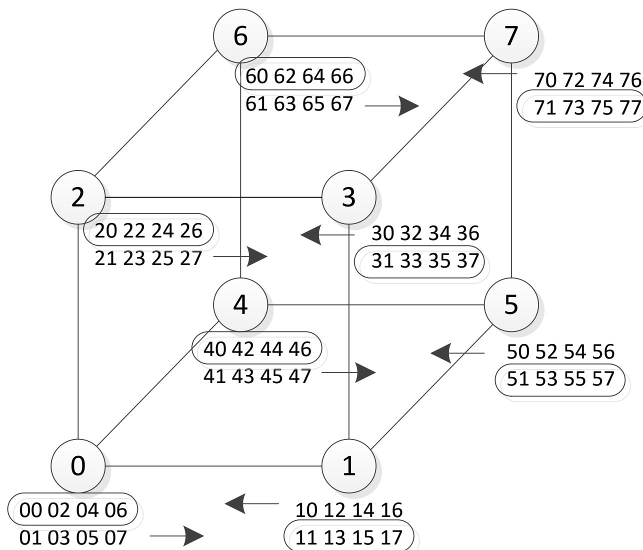
$t_{SF}^{M} = \left(\sqrt{N} - 1\right)(2t_i + Nt_w m)$

То же, что и для кольца, но $\sqrt{N} - 1$ передач., но 2-а раза.

Для гиперкуба:

Передаем по граням:

Первый шаг (по мл. координате)



Второй шаг: 0↔2, 1↔3 и т.д.

Потом по третей координате.

$t_{SF}^{HC} = (t_i + \frac{N}{2} t_w m) log_2 N$

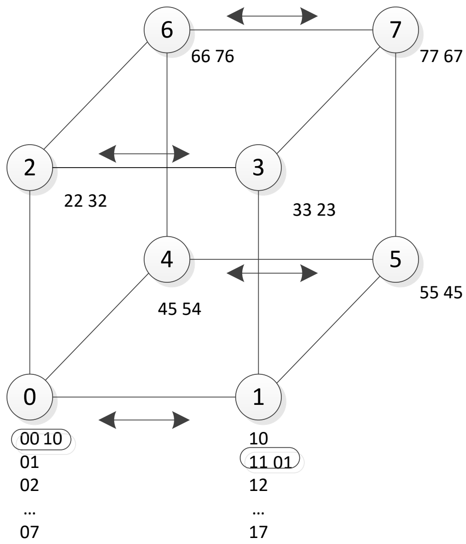
CT – для кольца и mesh-топологии не эффективно из-за коллизий. Для гиперкуба – возможно.

Лекция №6 (продолжение)

1. procedure ALL_TO_ALL_HC_CT (my_id, d)
2. begin
3. for i = 1 to 2^d – 1 do
4. begin
5. partner :- my-id XOR i
6. send $M_{my\text{-}id}$, partner$_{\text{-to}}$ partner
7. receive Mpartner, my-id to –pavku
8. end for
9. end procedure ALL_TO_ALL_HC_CT

Пример:

Первый шаг:

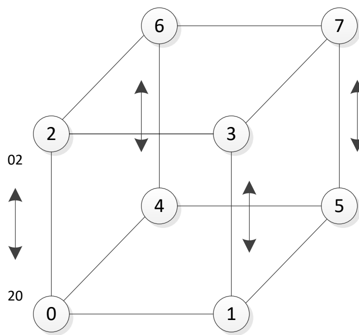


Фактически все-всем с персональным назначением – транспонирование матрицы.

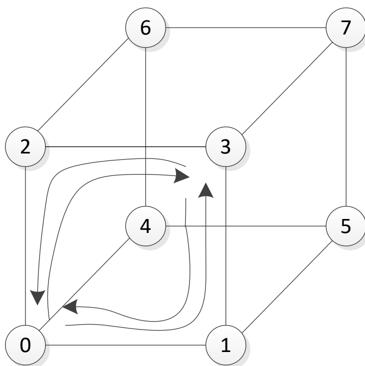
my_id = 000 $\frac{+\begin{smallmatrix}000\\001\end{smallmatrix}}{001}$

parther = 001

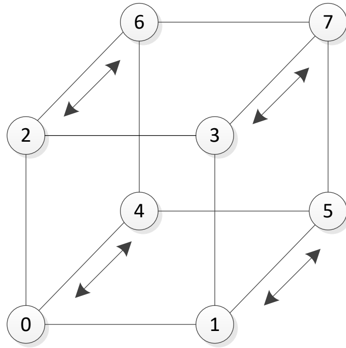
Второй шаг:



Третий шаг:

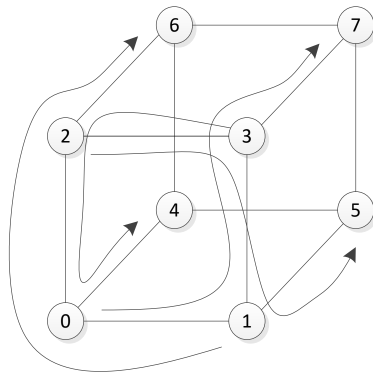


Четвертый шаг:



...

Седьмой шаг:

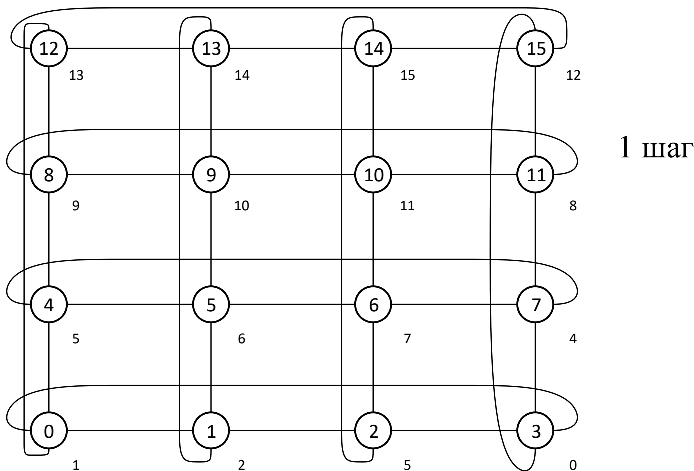


Все маршруты:
0 → 1 → 3 → 7
1 → 0 → 2 → 6
2 → 3 → 1 → 5
3 → 2 → 0 → 4
4 → 5 → 7 → 3
5 → 4 → 6 → 2
6 → 7 → 5 → 1
7 → 6 → 4 → 0

$t_{CT}^{HC} = (t_i + t_w * \mathrm{m})(N-1) + \frac{1}{2} t_T N \log_2 N$ ← лучше
$t_{3F}^{HC} = (t_i + t_w * \mathrm{m} * \mathrm{N}) * \log_2 N$

Маршрутизация на основе сдвигов

Речь идёт о регулярных топологиях (кольцо, куб….)



1 шаг

Три шага:

$1 \to q\left(mod\sqrt{N}\right)$ сдвиг

$2 \to$ поправочный шаг

$3 \to \left[\frac{q}{\sqrt{N}}\right]$

Пример #1 – failed

q = 5

1. 5 mod 4 = 1
2. поправочный шаг
3. $\left[\frac{5}{4}\right] = 1$ – не-не!

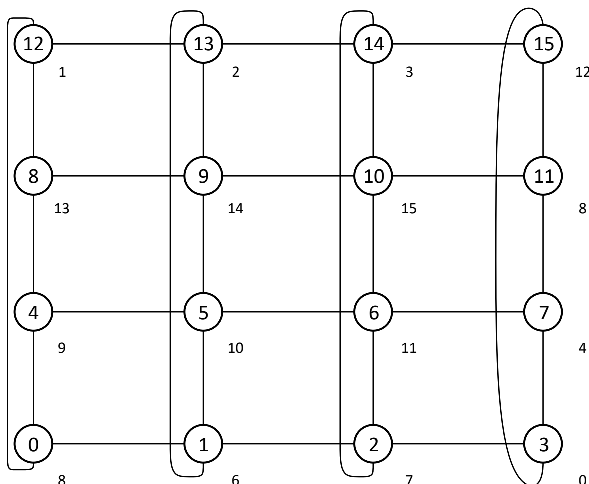
Пример #2:

q = 7 – выбир

1. 7 mod 4 = 3 (или на 1 влево)
2. поправочный шаг
3. $\left[\frac{7}{4}\right] = 1$ (вверх)

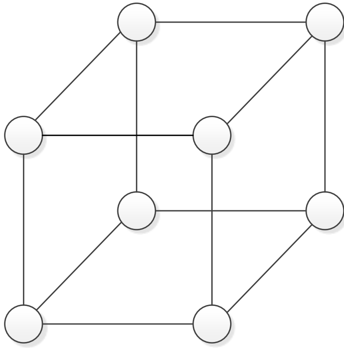
2 шаг (поправочный)

(на 3 строки вверх (или вниз))



$$t_{SF}^{M} = (t_i + t_w \mathrm{m})\left(2\left[\frac{\sqrt{N}}{2}\right]\right)$$

Гиперкуб:



Для q=7

$t_{SF}^{HC} = (t_i + t_w * \mathrm{m})(2\log_2 N - 1)$

0 – 1

1 – 2

2 – 3

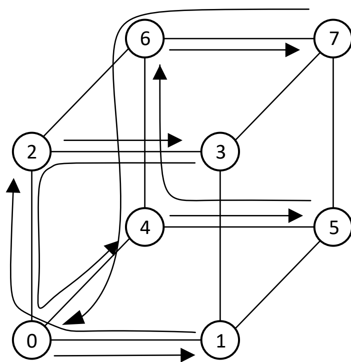
3 – 4

…..

Вариант 1

1 сдвиг

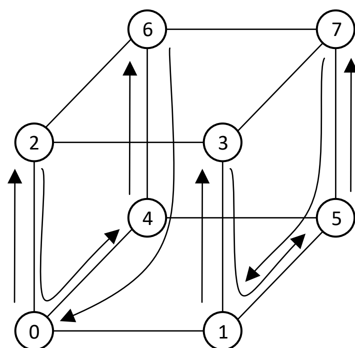
Всё выполняется параллельно и одно другому не мешает



Это конверсия маршрутиз.

Все выполняется параллельно.

Для сдвига 2:



0 – 2

1 – 3

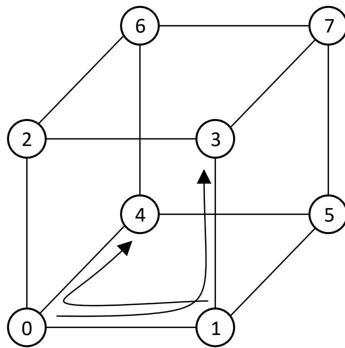
2 – 4

3 – 5

….

Вар. 2. 2 сдвига

Для сдвига 4:



Вар. 3 сдвиг 4

0 – 4

1 – 5

2 – 6

3 – 7

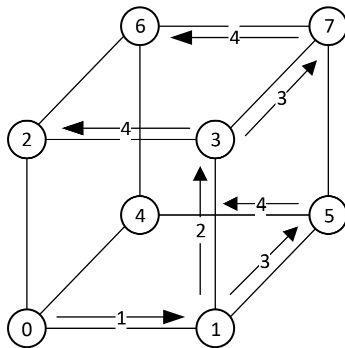
4 – 0 и т.д.

$t_{SF}^{HC} = t_i + t_w \mathrm{m} + t_T(\log_2 N - f(g))$

t(q) = i, где i при $\frac{q}{2^i}$ получ. целое число

Параллельная маршрутизация

Предполагается, что все сообщения находятся в одном узле.



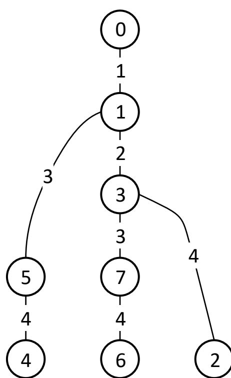
шаг 1: 0 – 1

шаг 2: 1 – 3(переход)

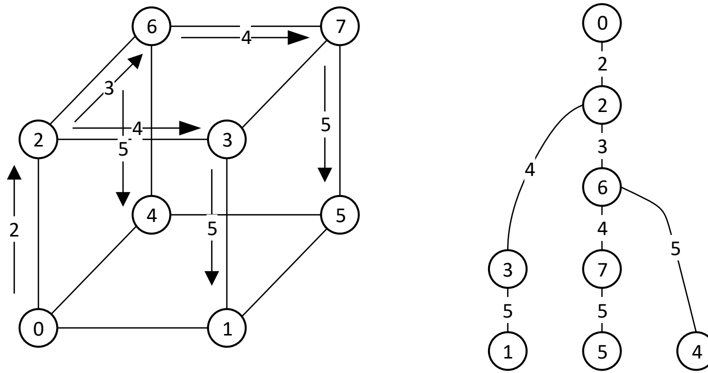
шаг 3: 1 – 5, 3 – 7

шаг 4: 1 - 0, 5 – 0, 3 – 2, 7 – 0

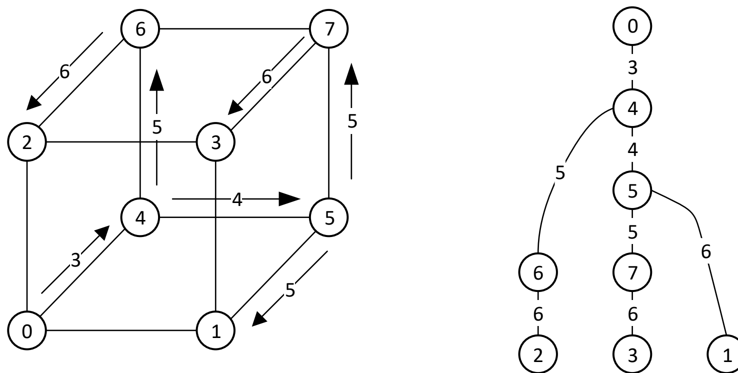
! нельзя из одного узла одновременно передавать



Дальше идёт второй поток:



3-ий поток:



$$t_p^{HC} = 2\left(t_i + t_w \frac{m}{\log_2 N}\right)\log_2 N = 2\left(t_i \log_2 N + t_N m\right)$$

Вопросы отказоустойчивости

Различают:

- неисправность (failure)

- ошибку (error)

- отказ (fault)

Неисправность обычно связана выходом из строя некоторого элемента системы на программном или аппаратном уровне.

Ошибка - это проявление неисправности в виде получения неправильного результата вычислений.

Отказ - некорректное выполнение функций, возложенных на систему.

Таким образом неисправность - это внутреннее состояние системы, которое характеризуется выходом из строя элемента.

Ошибка имеет информационный аспект - несет информацию о неправильных результатах.

Тогда отказ имеет внешний аспект, при котором пользователь узнаёт о некорректном поведении системы. Таким образом имеем латентное (скрытое) время неисправности,

которое определяет промежуток времени между моментом возникновения и получения неправильных результатов, а также патентное время ошибки.

То есть время между возникшим неправильным результатом и их обнаружения пользователем.

Поэтому система должна обладать средствами для локализации ошибок и средствами защиты системы от их распространения – т.е. средствами снижения нежелательных эффектов при распространении этой ошибки.

Кроме того, система должна обладать средствами для исследования дефектных элементов и замены их другими элементами, т.е. система должна обладать возможностью восстановления работоспособности системы.

Для обеспечения отказоустойчивости можно использовать: грубый подход, тонкий подход и введение избыточности.

Грубый подход заключается в том, чтобы исключать неисправные элементы с сохранением топологических свойств.

Тонкий подход – если вышел элемент из строя, то сделать алгоритм, который будет обходить неисправный элемент. Тут минимальные потери, но алгоритм существенно усложняется.

Основанные на избыточности – создание дополнительных связей и вершин.

Содержание

Понятие сложности алгоритмов -- 1

Метрика вычислительных систем --- 1

Синтез топологий на основе лат. Квадрата ------------------------------------- 8

Синтез топологий на основе кодовый преобразований --------------------------- 10

Синтез топологий на основе лин. уравн. --- 17

Закон Амдахла -- 19

Закон Густавсона -- 19

Отображение различных топологий в гиперкубе ---------------------------------- 20

Алгоритмы маршрутизации -- 22

Алгоритм one-to-one -- 23

Алгоритм one-to-all -- 24

Алгоритм all-to-all -- 28

Алгоритм one-to-all-personalized -- 32

Маршрутизация на основе сдвигов --- 37

Параллельная маршрутизация --- 39

Вопросы отказоустойчивости --- 41

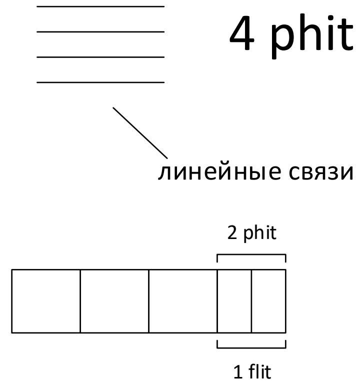
Коммуникационные среды -- 42

Коммутация каналов -- 44

Коммутация пакетов -- 45

Виртуальная коммутация пакетов -- 46

Всё это вызывает временные и технические затраты.
Будем различать различные объёмы информации, которые являются управляемыми. Обычно различают минимальную логическую единицу и минимальную физическую единицу. Именно минимальная логическая единица определяет минимальную единицу управления. Используется понятие «flit» для логической минимальной единицы и «phit» для физической.



Мы будем предполагать (для простоты), что 1 phit = 1 flit.
Выделим задержки:
t_r ($t_{routing}$) – время для решения вопросов маршрутизации (определения маршрута);
t_s (t_{switch}) – задержка коммутатора;
t_w – передача информации.

L – длина пакета.
W – количество битов в одном flit.
L+W – длина сообщения.

Частота коммутирующих элементов B Гц. Тогда пропускная способность: $B * W$. Задержка в канале: $\frac{1}{B}$.

Коммутация каналов

$t = t_{setup} + t_{data}$

t_{setup} – уст. связи, t_{data} – передача сообщения

D – диаметр

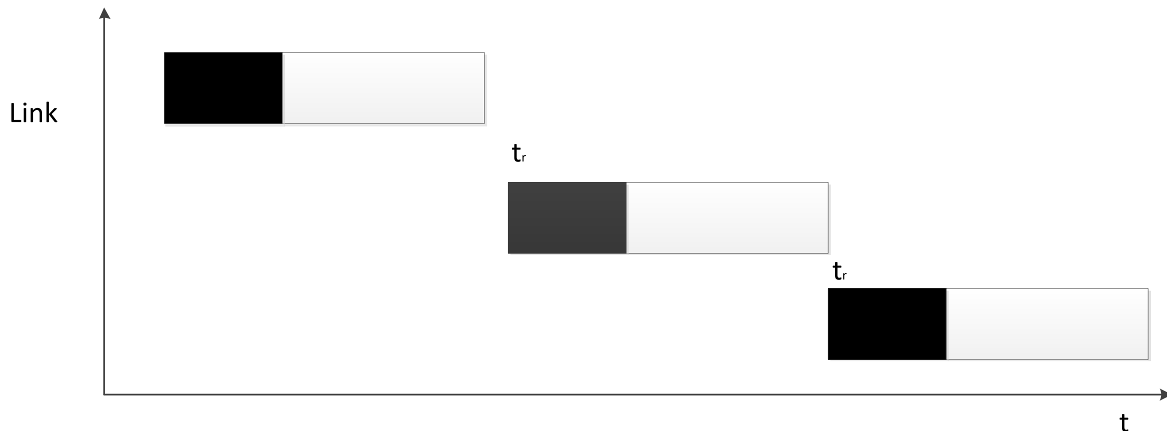
d_{ij} – расстояние между i, j

$t_{setup} = d(t_r + 2\ (\ t_s + t_w))$

$t_{data} = \frac{1}{B}\frac{1}{W}$

Коммутация каналов может быть эффективна, если сообщения добавляются длинными и возникают они довольно редко. Основные недостатки такой сети - возможность блокирования длительный период времени во время взаимодействия с другими абонентами. Основное преимущество такой системы связи является отсутствие использования буферов для передачи сообщений.

Другой способ — коммутация пакетов



В отличие от предыдущего варианта для передачи информации в каждый момент времени используется только один роутер, что практически исключает блокировку других абонентов. Но здесь в обязательном порядке требуется использование входных и выходных буферов в каждом из промежуточных маршрутизаторов. При чем объем буферов должен быть равен размеру пакетов.

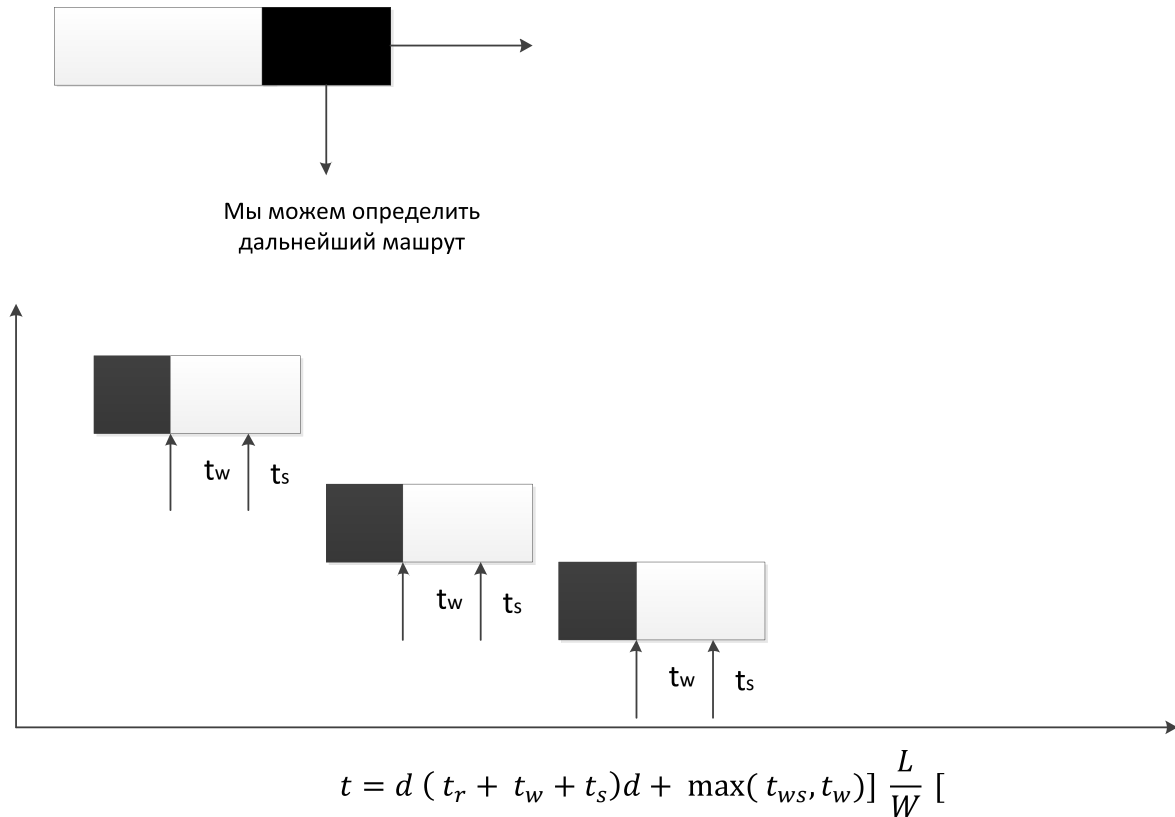
$$t = d\ (\ t_r +\ t_w + t_s)]\ \frac{L+W}{W}\ [$$

Данная коммутация может быть эффективна в том случае, когда сообщения короткие и приходят довольно часто.

Виртуальная конвейерная коммутация пакетами.

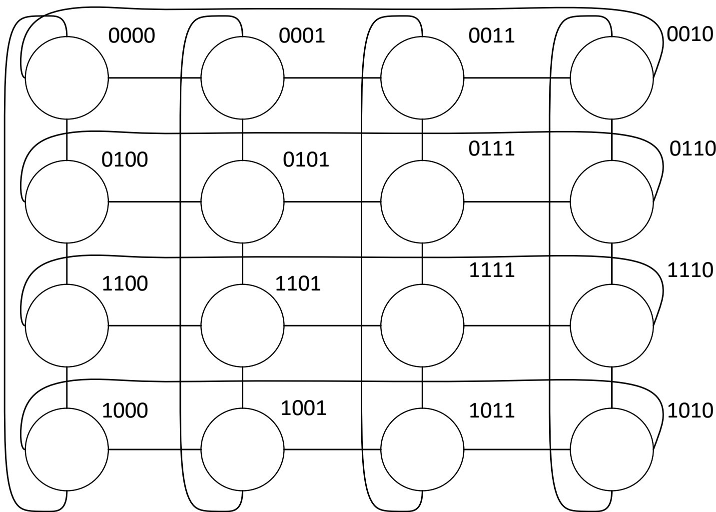
Предполагает, что заголовок идет первым в пакете и занимает ограниченное количество байтов(например, 4 байта). Следовательно, через период времени равный передачи 4 байтов. Мы можем определить маршрут передачи информации, что позволяет без буферизации всего пакета последовательно по flit`ам передавать его в спец. Router. При этом выходные буферы могут вообще не использоваться.

Это означает:



$$t = d\,(t_r + t_w + t_s)d + \max(t_{ws}, t_w)]\,\frac{L}{W}\,[$$

Возьмем mesh-топологию:



При решении вопросов отображения одной топологии в другую могут иметь место 3 варианта:

1. Несколько ребер исходной топологии могут быть представлены одним ребром другой топологии. Это называется сужением топологии и обычно приводит к дополнительным очередям на передачу информации.

2. Одно ребро исходной топологии может представляться с помощью нескольких ребер в отображаемой топологии (растяж. топологии, которое как правило приводит к увеличению времени затрат на передачу информации)

3. Одна вершина исходной топологии может представляться с помощью нескольких вершин отображаемой топологии. Здесь могут быть последствия: «+» - возможность распараллелить обработки

«-» - увеличивается время передачи информации

Лабораторная работа

Придумать 5 нестандартных топологий (должны масштабироваться)

1. Описать формально топологию (матр. смежности)
2. Наращивать программно
3.

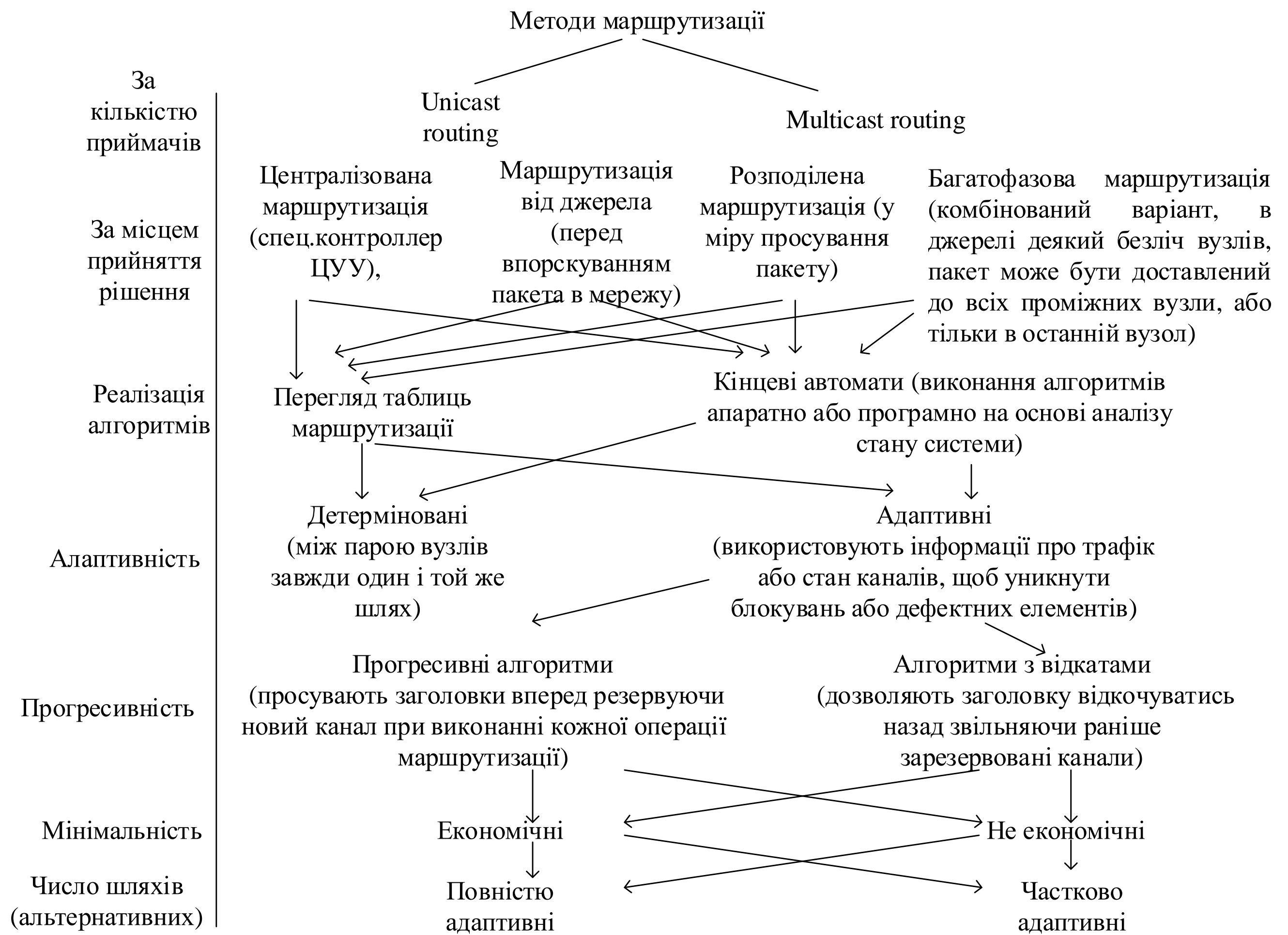
N	D	$\bar{D}$	C	S	P
5 10 15 … ~100					

C = S*D*N

Протокол: топология, законы масштабирования, листинг, результат (табл.)

1. Линейка
2. Кольцо
3. Дерево
4. Решетка
5. ? (линейка)

Методы маршрутизации



Алгоритм маршрутизации классифицируется по нескольким критериям:

1) <u>По числу приемников</u>. Если пакеты предназначаются для одного приемника, то такую маршрутизацию называют unicast, а если пакеты предназначены для нескольких приемников, то маршрутизацию называют <u>multicast</u>.
2) <u>По месту принятия решения о маршруте</u>. Если для этой цели используется специально реализованный центральный контроллер, то такое принятие решений определяет <u>централизованную маршрутизацию. (centralized routing</u>). Если принятие решений осуществляется перед впрыскиванием (вводом) пакетов в сеть, то такая маршрутизация называется source routing (маршрутизацией в источнике) или распределенным способом. Если же решение принимается по мене прохождения пакета через сеть, то такая маршрутизация называется распределенной (distributed routing). При этом возможны и комбинированные решения. Одно из таких решений основано, на многофазовой маршрутизации (multiphase routing). При ее использовании источник вычисляет некоторое множество узлов назначения, а маршрут между ними определяется распределенным способом. Пакет может доставляться во все вычисленные узлы назначения (multicast routing) или только в последний узел назначения (unicast routing). При этом промежуточные узлы используются таким образом, что бы избежать блокировки или дефектных узлов.

Алгоритмы маршрутизации могут быть реализованы различными путями. Наиболее распространенные пути, рассматриваемые выше, предполагают просмотр таблиц маршрутизации (table-looking) или предполагает выполнение алгоритмов, реализованных

аппаратно или программно, на основе определения состояния системы (finite-state machine). В обоих случаях алгоритмы могут быть децентрализованными или адаптивными. При детерминированной маршрутизации между парой узлов всегда имеет место один и тот же путь. Адаптивные алгоритмы используют информацию о сетевом трафике и состоянии каналов и определяют путь таким образом, что бы избежать блокировок и дефектных элементов. Адаптивные алгоритмы могут классифицироваться в соответствии с их поступательным характером. При этом следует выделить прогрессивные элементы (progressive) и алгоритмы с откатами (back hacking). Прогрессивные алгоритмы маршрутизации продвигают заголовок вперед, резервируя новый канал при выполнении каждой операции маршрутизации. Алгоритмы с откатами позволяют заголовку откатываться назад, освобождая ранее зарезервированные каналы. Алгоритмы с откатами используются главным образом для отказоустойчивой маршрутизации.

Алгоритмы маршрутизации могут классифицироваться в соответствии с их экономичностью, как экономичные так и не экономичные. Экономичные алгоритмы при маршрутизации используют только те каналы которые доставляют пакет к их приемникам, их называют минимальными. Не экономичные алгоритмы могут так же использовать и те каналы, которые не ведут пакеты приемникам, их называют не экономичными(не минимальными)

Алгоритмы маршрутизации могут так же классифицироваться в соответствии с числом альтернативных путей как полностью адаптивные (fully adaptive) так и частично адаптивные (partially adaptive).

Рассмотрим unicast-routing алгоритмы. Централизованная маршрутизация предполагает использование центрального устройства выполнения. Этот вид маршрутизации используется в системах SIMD. В случае маршрутизации в источнике узел-источник определяет маршрутную информацию на основе неблокирующих алгоритмов. Вычисленный путь храниться в заголовке пакета, который используются в промежуточных узлах для резервирования каналов. Алгоритмы маршрутизации могут использовать только адреса текущих узлов и узлов назначения, что бы вычислить путь (детерминированная маршрутизация) или может так же использовать информацию, взятую от других узлов о условиях трафика (адаптивная маршрутизация). Заметим, что получение информации от других узлов сопряжено со значительными накладными расходами. К тому же, информация может оказаться устарелой. Таким образом, адаптивная маршрутизация представляет интерес только тогда, когда трафик меняется очень медленно. Маршрутизация в источнике используется главным образом в нерегулярных топологиях. Myrinet поддерживает нерегулярные топологии и использует маршрутизацию в источнике. Первые несколько флитов заголовка пакета содержит адрес портов коммутатора промежуточных коммутаторов.

Маршрутизация в источнике может использоваться для мультикомпьютерных взаимодействий сетей. Условие трафика могут изменяться быстро в этих сетях, а следовательно, адаптивная маршрутизация не представляет интересов. Так как заголовок пакета должен передаваться через сеть и использовать сетевую пропускную способность, весьма важно минимизировать длину заголовка.

Одним из методов минимизации заключается в так называемой маршрутизации уличных знаков(street-sign). Заголовок в данном случае является набором указателей для водителя в городе. Необходимо знать только названия улиц, где водитель должен повернуть, и направление поворота. Более того, пакет, прибывающий в промежуточный узел по умолчанию выбирает выходной порт в том же измерении и направлении как и в текущем канале. Для каждого поворота заголовок должен содержать адрес узла, в котором будет иметь место

поворот, и направление поворота. Кроме того, эта информация должна храниться в заголовке в соответствии с порядком, в котором узлы достигаются. При получении флита заголовка маршрутизатор сравнивает адрес в флите с адресом локального узла. Если они совпадают, пакет либо должен быть передан с поворотом, либо достигает приемника (4.1).

Для повышения эффективности большинство аппаратных маршрутизаторов используют распределенную маршрутизацию. При конвейерной маршрутизации маршрут пакета определяется только при достижении заголовком некоторого узла. При этом заголовок является очень компактным, то есть требует наличие адреса назначения и нескольких контрольных битов. При распределенной маршрутизации каждый промежуточный узел должен принимать решение о маршруте на основе локальных сведений о сети. На основе повторения этого процесса в каждом промежуточном узле пакет должен достигать узла назначения. Заметим, что алгоритмы маршрутизации в данном случае выполняются без наличия глобальных сведений о сети. Это может достигаться с учетом того, что разработчик знает топологию всей сети.

Распределенная маршрутизация используется в основном в регулярных топологиях так, что на всех узлах могут выполнятся один и тот же алгоритм. Почти все коммерческие топологии состоят из нескольких ортогональных измерений. Поэтому в них легко вычислять расстояния между текущим узлом и узлом назначения как сумму сдвигов (шагов) во всех направлениях. В таких топологиях принятие решения намного проще чем в других топологиях.

В многофазовых unicast routing в источнике вычисляется некоторый промежуточный узел. Пакет передается в этот узел на основе распределенной маршрутизации. После принятия пакета промежуточный узел вновь впрыскивает пакет в сеть, используя другой промежуточный узел или финальный узел для пакета. Примером многофазовой маршрутизации может служить алгоритм «наугад». В этом алгоритме в источнике «наугад» вычисляется промежуточный приемник. После принятия пакета, промежуточный узел направляет пакет по направлению финальному приемнику. Алгоритм случайной маршрутизации «наугад» был предложен с целью понижения уровня конкуренции на основе рандомизации (угадывания) маршрутов следования наборов пакетов, передаваемых между каждой парой «источник-приемник». Однако случайный маршрут разрушает все коммуникационные области, существующие в сетевом трафике. Многофазовая маршрутизация предложена так же для реализации безотказной маршрутизации. В этом случае промежуточные приемники используются что бы снять (разрушить) зависимости между каналами и избежать блокад при наличии отказов.

Независимо от того, где алгоритм маршрутизации вычисляется, он должен обеспечить доставку пакета до приемника. В параллельных системах разработчик выбирает топологию связи в сети. Часто выбранная топология может быть декомпозирована на несколько измерений (mash, гиперкуб, тор). В таких топологиях возможно использование простых алгоритмов маршрутизации, основанных на определении состояния системы подобно e-cube (dimension-order routing). Это маршрутизация продвигает пакеты до перехода к другому измерению (до поворота) в возрастающем порядке, пока не станет нулевой разницы в данном измерении между адресом назначения и адресом приемника. Это метод уличных знаков. Маршрут при этом может определятся в источнике, но может определятся и распределенным способом. Тогда в каждом промежуточном узле алгоритм маршрутизации снабжает выходной канал информацией о повороте на следующее измерение в случае нулевого значения рассмотренной разницы.

В случае использования нерегулярных топологий, которые оказываются необходимыми пользователю обычно используются маршрутизация табличная. При этом такая таблица маршрутизации должна быть в каждом узле и обладать числом входов, равным количеству узлов в сети. Опять таки маршрутизация может осуществляться либо в источнике, либо в каждом промежуточном узле. В первом случае адрес приемника определяет соответствующий вход в таблицу и определяет полный маршрут для достижения приемника. Во втором случае каждый вход в таблицу определяет выходной канал для перехода на следующий узел, ведущий к приемнику. Однако такой подход приемлем для малых систем, так как размер таблицы возрастает линейно с увеличением числа узлов. Рассмотрим наиболее популярные алгоритмы маршрутизации, которые хорошо проработаны методически, являются простейшими и используются в топологиях, которые можно декомпозировать по нескольким ортогональным измерениям.

Простейшие из них называются размерно-упорядоченной маршрутизацией (dimension-order routing). Он основан на выполнении алгоритма в строго возрастающем порядке измерения и понижении расстояния между промежуточным узлом и узлом назначения до нуля, после чего происходит переход на другое измерение.

В этих алгоритмах

FirstOne – функция которая возвращает положение первого бита набора, равного 1.

Internal – канал, связанный с локальным узлом.

Отказоустойчивые системы

Отказоустойчивые системы - это такие системы, которые позволяют продолжать корректное выполнение задач при появлении неисправностей в аппаратных или программных средствах системы. Отказоустойчивость – это способность системы обрабатывать внутренние неисправности без прерывания функционирования системы.

Будем различать три базовых термина, связанных с отказоустойчивостью: неисправность-дефект (fault), ошибка (error) и отказ (failure, crash). Часто наряду с термином отказ используется термин сбой (malfunction), однако первый из них является более общим, поскольку связывается и аппаратными и программными средствами системы.

Неисправность-дефект – это изъяны или повреждения аппаратных или программных средств. Таким образом, неисправность – это выход из строя или дефект компонентов аппаратных или программных средств.

Ошибка – это результат или проявление неисправности. Иными словами, ошибка – это отклонение от точности и корректности вычислений. Если имеет место такая ситуация, то она приводит к отказу.

Отказ – это не выполнение некоторых действий, которые должны были выполнится или ожидаются. Отказ – это так же выполнение некоторых функций в условиях ограниченного качества или количества.

Отказ – это не выполнение или не корректное выполнение некоторых функций, возложенных на систему. Кроме того, отказ - выполнение некоторых функций в условиях ограниченного качества или количества.

Таким образом, ошибка имеет информационный аспект и связана с получением неправильного информационного блока.

Тогда отказ имеет внешний (пользовательский) аспект, на основе которого пользователь обнаруживает неправильную работу системы.

Здесь имеет место два интервала времени:

1) Латентное (скрытое) время неисправности между моментом возникновения неисправности и ее проявлением в виде ошибки
2) Латентное время ошибки – между моментом появления ошибки и обнаружением ее пользователем

Для реализации отказоустойчивости система должна обладать средствами для обнаружения и локализации неисправностей. Кроме того, должны быть предусмотрены средства для изоляции неисправности или для сдерживания эффектов ее распространения.

Система должна обладать средством для реализации восстановительных работ в системе с целью восстановления ее работоспособности.

Fault-tolerance

(отказоустойчивость, основанная на избыточности)

Технология, основанная на введении избыточности.

Концепция избыточности предполагает введение избыточной информации, ресурсов или времени. При этом различают аппаратную, программную, информационную и временную избыточность. Часто различные виды избыточности необходимо сочетать.

Аппаратурная избыточность

Физическое дублирования – это наиболее общая форма избыточности в системах.

Будем различать три основные формы аппаратурной избыточности.

Пассивная избыточность использует технику маскирования отказов, скрывая случаи отказов и предотвращая результаты от ошибок. Таким образом пассивная избыточность реализует безотказные системы без выполнения каких-либо действий со стороны системы или оператора.

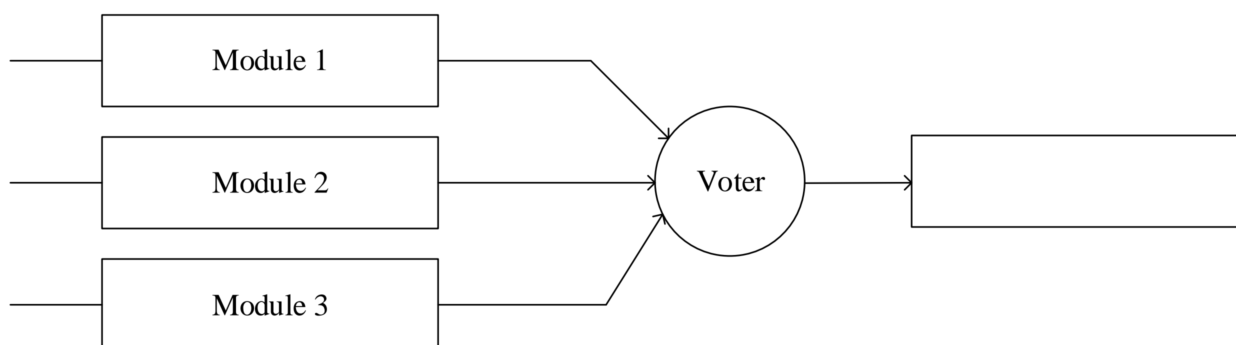
Активная избыточность или, иными словами динамическая избыточность отказоустойчивость в данном случае достигается на основе выделения отказов и выполнения соответствующих действий по устранению отказавших узлов из системы на основе реконфигурации. Данная избыточность предполагает возможности системы к выдилению ошибок, их локализации и восстановление работоспособности системы.

Гибридная избыточность сочетает в себе наиболее привлекательные свойства обоих предыдущих подходов

Пассивная избыточность

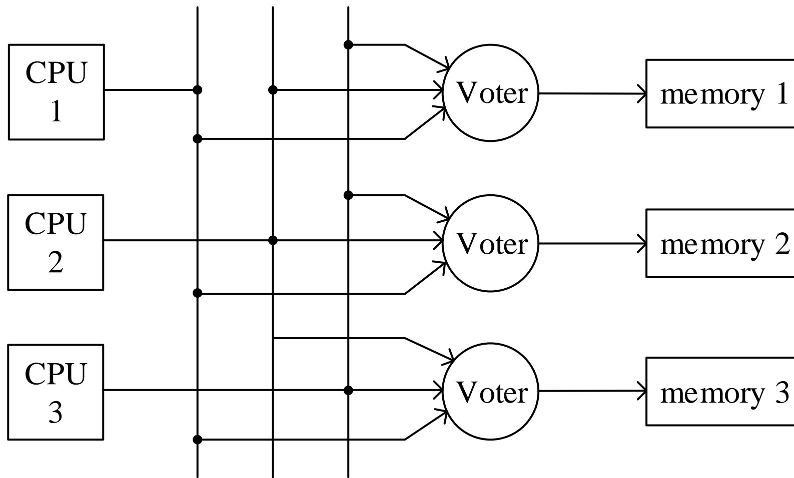
Основана на использовании схемы голосования (voter), благодаря которой появляется возможность маскирования возникающих отказов. Большая часть подходов данного направления основана на концепции мажоритарного голосования, то есть наличия мажоритарного элемента.

Большинство форм реализации данного подхода основаны на тройной модульной избыточности (TMR)



single-point-of-failure

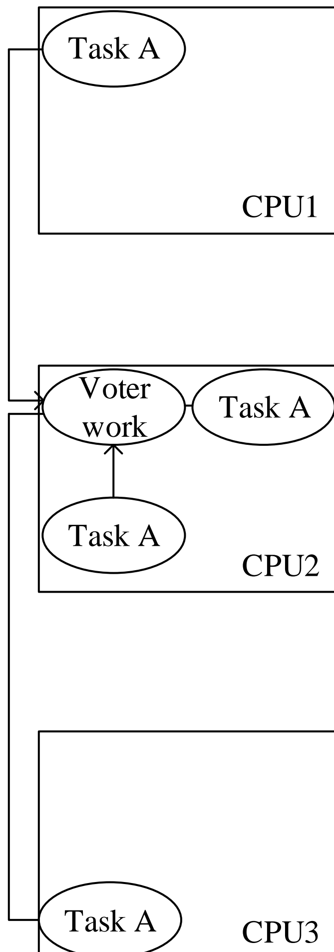
Однако если Voter неисправен, вся система неработоспособна. Надежность системы не может быть выше надежности Voter. Для преодоления этой проблемы может использоваться следующая схема.



Restoring

5 CPU → 2 ошибки можно изолировать

Пассивная избыточность: voter может быть реализован не только аппаратно, но и программно

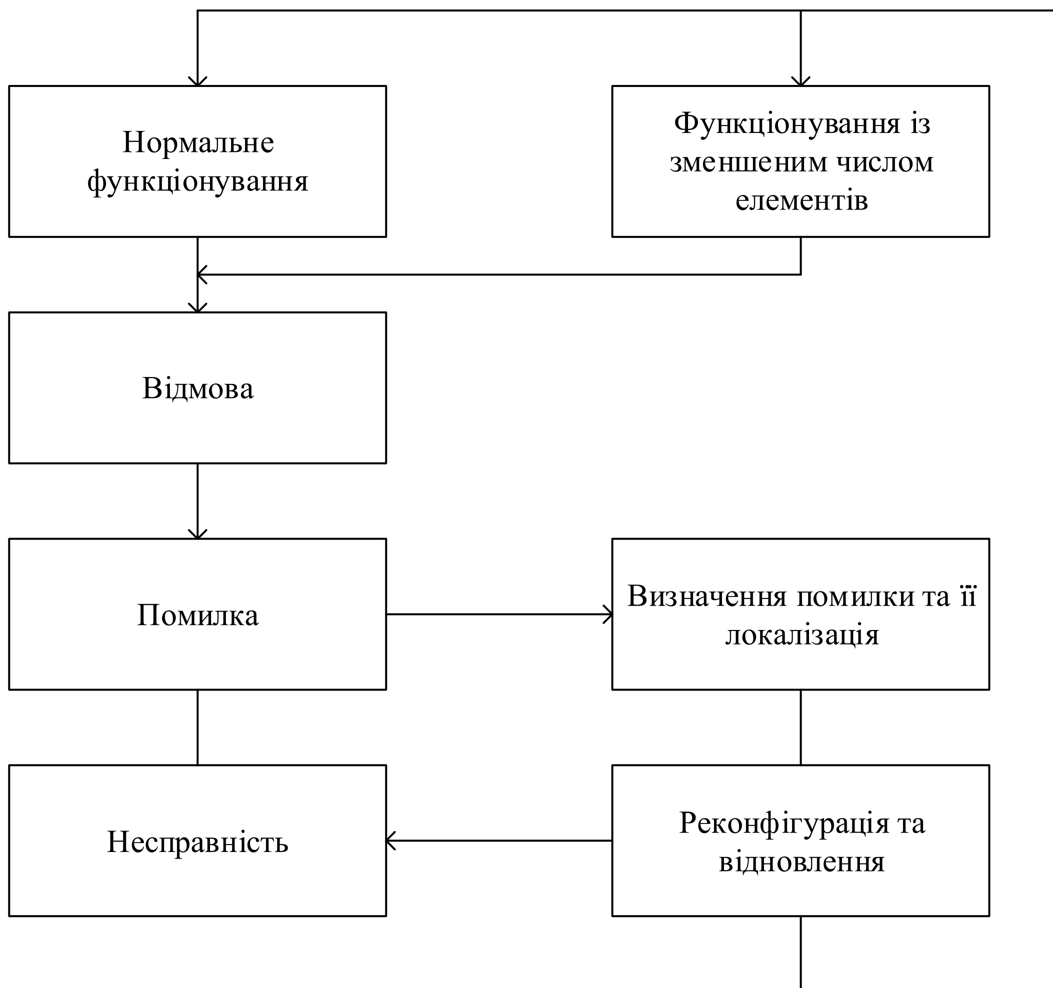


Скорость меньше, но гибкость выше

Проблемы согласования сигналов

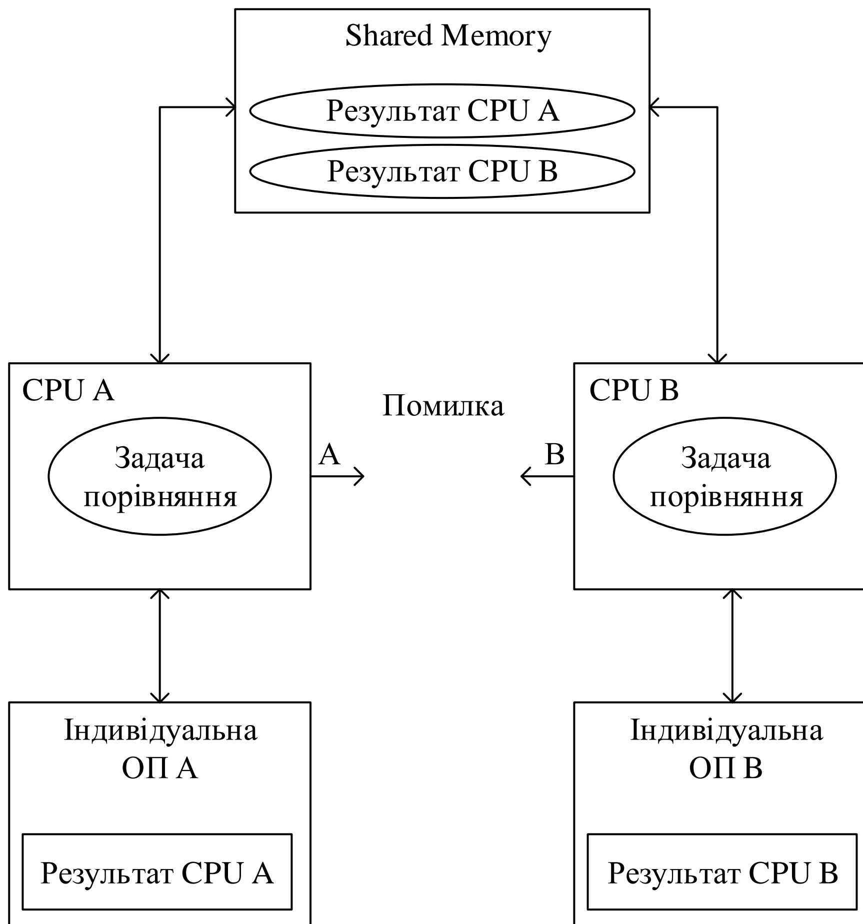
Активная избыточность

Динамическое введение избыточность



Один из методов выделения отказа основан на простом дублировании схем различной сложности и выполнении операции сравнения. Очевидно, что дублирование не может выделить правильный результат, но выделить несоответствие между результатами, а следовательно, найти ошибки, он может.

Принцип дублирования на рис.



Другая форма активной избыточности основана на резервной замене или резервированном замещении (standby replacement, standby sparing)

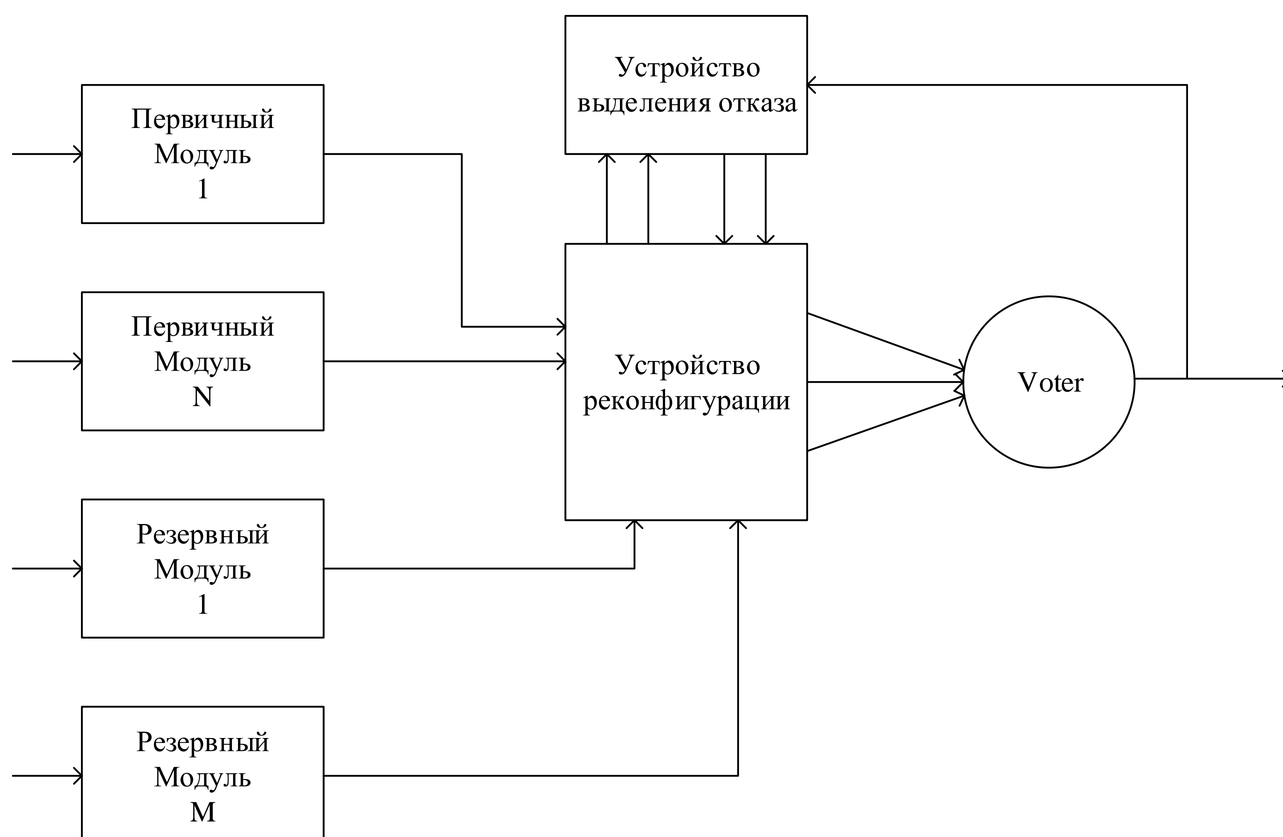
В данном случае один модуль функционирует, а другой находится в резерве. Здесь могут быть различные схемы.

Если отказ выделен и локализован, неисправный модуль удаляется и замещается на основе реконфигурации другим модулем.

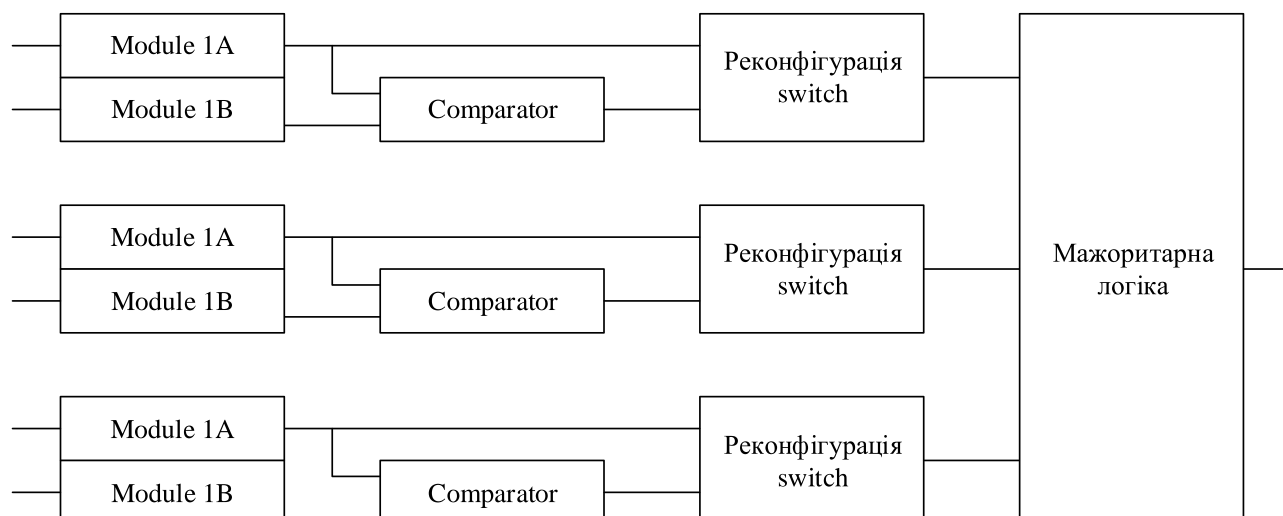
Резервная замена должна привести систему в действующее состояние, но при этом требуется краткосрочное разделение системы для реализации реконфигурации. Это разделение системы должно быть минимизировано. С этой целью используется горячее резервирование. Противоположность горячему резервированию, холодное резервирование требует включения модуля и его инициирование. Pair_and_a_spare – модули действуют в паре.

Гибридная организация избыточности

Есть несколько подходов к гибридной организации избыточности в системах. Один из них основан на N-модулярной избыточности (NMR) с резервированием.



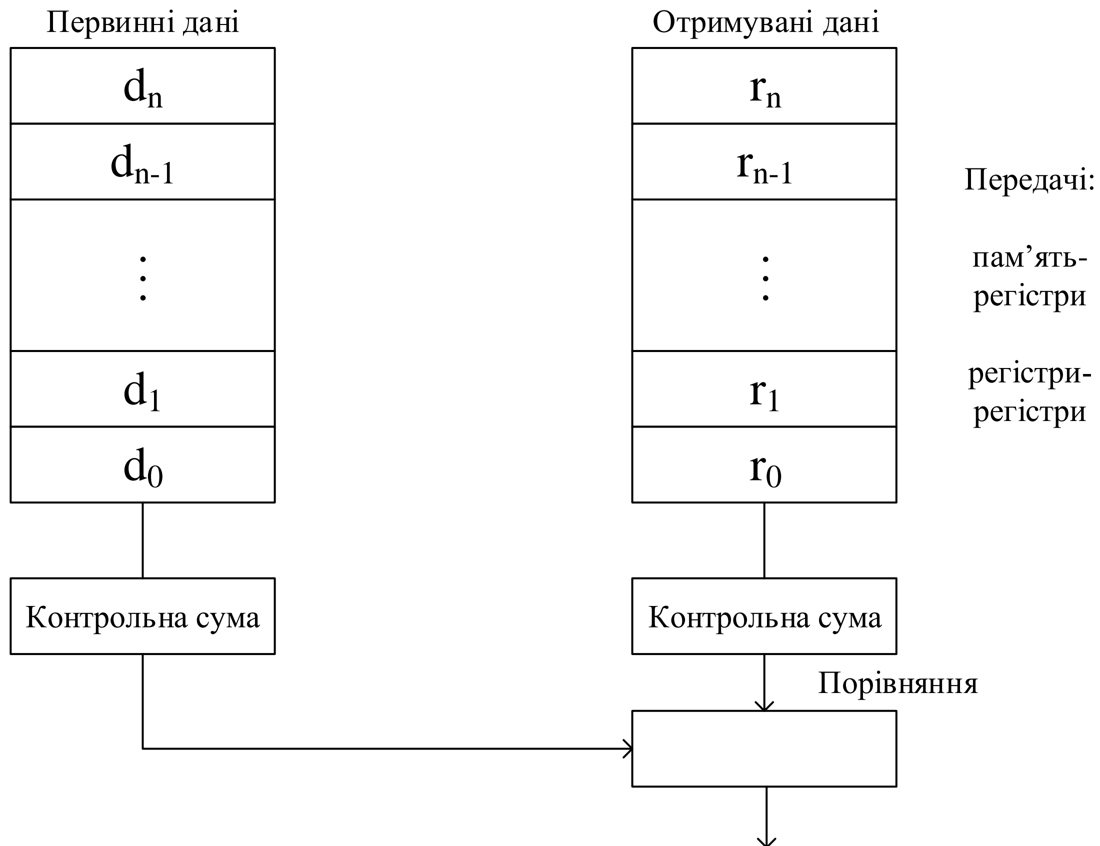
Следующий подход основан на тройном _ резервировании.



Сочетает дублирование со сравнением и мажоритарный подход. Компаратор осуществляет выделение ошибок и замену их резервными элементами. Далее мажоритарная логика.

<u>Информационная избыточность</u>

1) Коды с контролем по честности
2) m из n коды
   Длина кода n
   m 1 должны содержать
   Если будет m+1 или m-1 – ошибка
3) Дублированные коды
   Дублирование первичной информации
4) Образование контрольной суммы
   РИС8



5) Циклические коды
6) _______ коды
7) Коды Бергера (Berger)
8) Коды Хемминга, ________
9) Самопроверяющиеся коды

Временная избыточность

Выделение временных, не постоянных ошибок

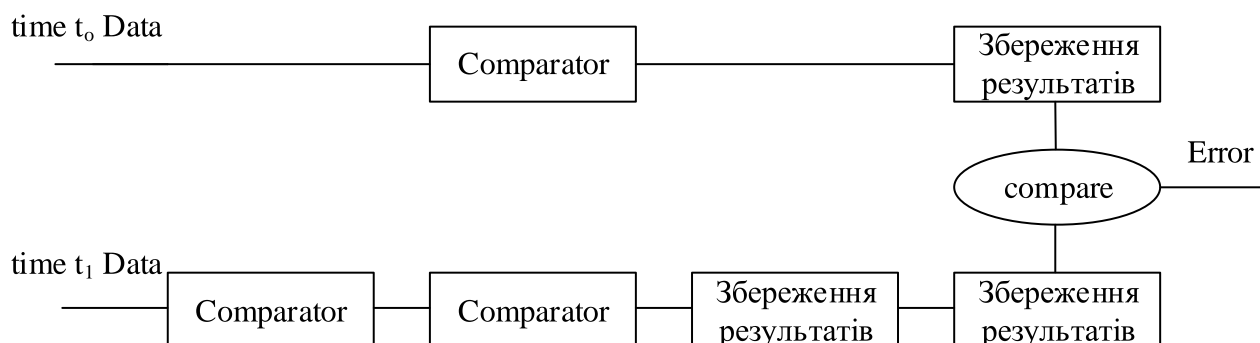
Transient (Временные)

Основанная концепция временной избыточности – это повторение вычислений таким образом, что бы выделить отказы. Таким образом необходимо два, три раза повторять вычисления и, если результаты не совпадают, то тем самым может быть выделена ошибка.

Но это касается случайных, неустановившихся ошибок. Для отказов установившихся данный механизм не работает.

Выделение постоянных отказов

Концепция на рис. 1.17



Программная избыточность

Различают следующие варианты программной избыточности:

Согласованная (целостная) проверка

Мандатная проверка

Программное дублирование

Согласованный контроль предполагает наличие априорной информации для верификации. Например может быть известно, что цифровая величина не должна превышать некоторые значения. Если сигнал превышает данную величину, то это говорит об ошибке.

Другой пример: для представления команды обычно используется в k разрядов из n, где $k<n$.Таким образом можно выделить множество недопустимых кодов, которые и будут свидетельствовать об ошибке.

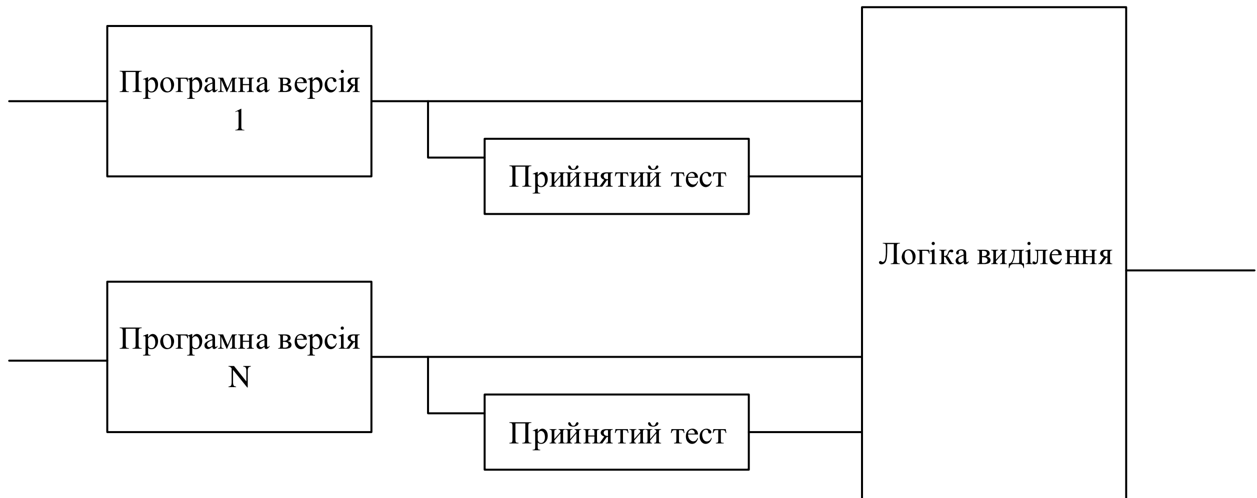
Мандатный контроль выполняется для проверки возможностей системы. Например, необходимо знать, является ли наша память приемлемой, или все ли процессоры в мультипроцессорной организации функционируют хорошо, или хорошо ли работает ALU.

a) Запись и чтение ______
b) Выполнение определенных команд на определенных данных и их сравнение с результатами, записанными в ROM
c) Проверка правильности функционирования каналов передачи данных.

N-самоконтролирующее программирование

Ошибки software связаны с некорректной разработкой или ошибками кодирования. Следовательно техника выделения ошибок в software должна выявлять изъяны разработки. Простое дублирование и сравнение не может выделить ошибки software, если модули software идентичны, так как ошибки будут появляться в обоих модулях.

Концепция самоконтролируемого программирования представлена на рис. 1.18



Идет чтение N уникальных версий и каждая версия включает в себя набор принятых тестов. Принятые тесты контролируют результаты, получаемые при выполнении программы на основе согласованной или мандатной проверки.

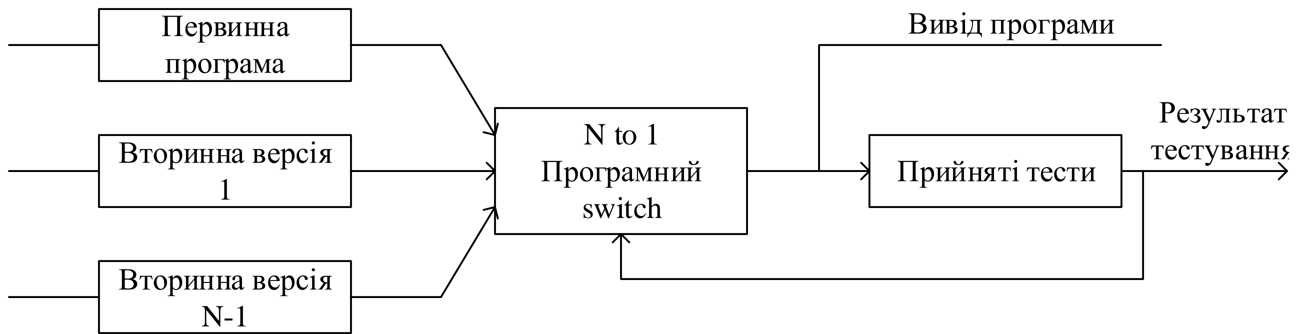
N-версное программирование

Основная концепция N-версного ____ программирования связана с разработкой и кодированием приемлемых модулей software. Основная концепция N-версного программирования связана с разработкой и кодированием модулей software N раз и мажоритарным выбором.

Каждый из модулей разрабатывается и кодируется отдельной группой программистов. Каждая группа разрабатывает software на основе одного и того же набора спецификаций так, что каждый модуль выполняет одну и туже функцию. Независимость программистов предполагает, что одна и та же ошибка не будет сделана. Количество ошибок может быть равно $\frac{N-1}{2}$

Восстанавливающиеся блоки

Аналогично холодному резервированию используется N версий программ и единственный набор тестов. Одна из версий программ рассматривается как первичная, а остальные N-1 версий разрабатываются как резервные. Первичная версия используется до тех пор, пока она успешно проходит принятые тесты. В противном случае начинает выполнятся первая вторичная версия. Этот процесс продолжается до тех пор, пока одна из версий будет успешно проходить тесты.



Информационная избыточность

Два подхода

1. Основан на использовании расстояния Хемминга
2. Основан на разделении данных

А) Код с контролем по четности (нечетности)

	P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	P_{11}	P_{12}	P_{13}	P_{14}	P_{15}	P_0	P_1	P_2	P_3
P_0	0				4				8				12				1	0	0	0
P_1		1				5				9				13			0	1	0	0
P_2			2				6				10				14		0	0	1	0
P_3				3				7				11				15	0	0	0	1

РИС

Реконфигурация в системах

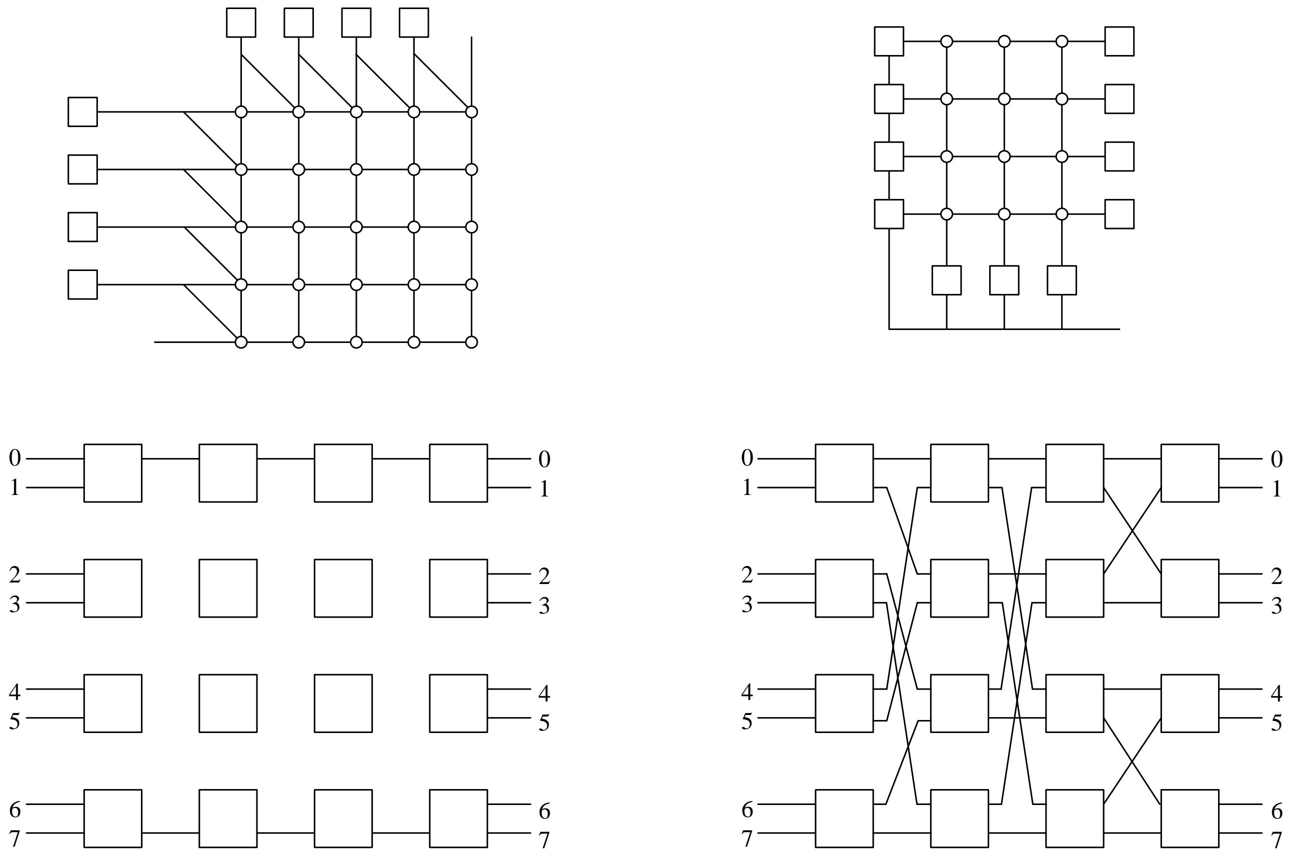
3 подхода:

1-й – грубая деградация производительности на основе существенного уменьшения числа элементов.

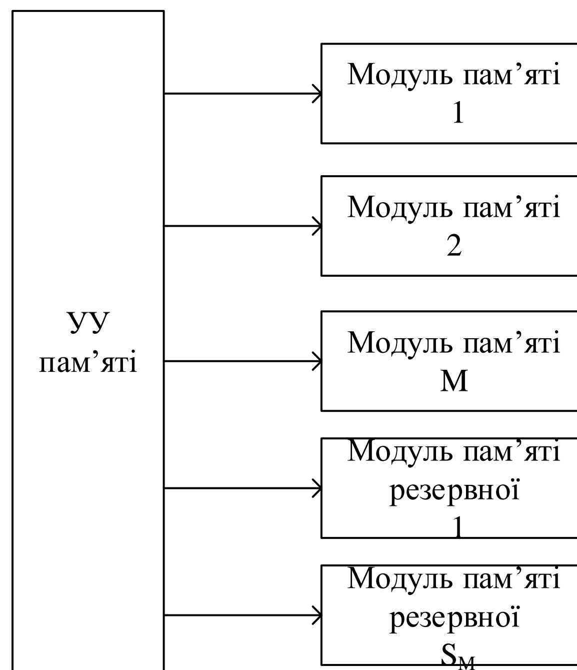
2-й - изящная деградация производительности на основе исключения отказавших элементов

3-й – введение резервных элементов

1) Шинно-ориентированная система
2) _______



Пример избыточности



Пример активной избыточности. Два уровня активной избыточности.

Память организована в M модулей, каждый объемом в 256 megabits (Mb)

1. Fault, error, failure (латентное время, отн., _____)
2. Отказоустойчивые системы, основанные на избыточности

   А) Пассивная, статическая избыточность

Б) Активная, динамическая избыточность

В) Гибридная

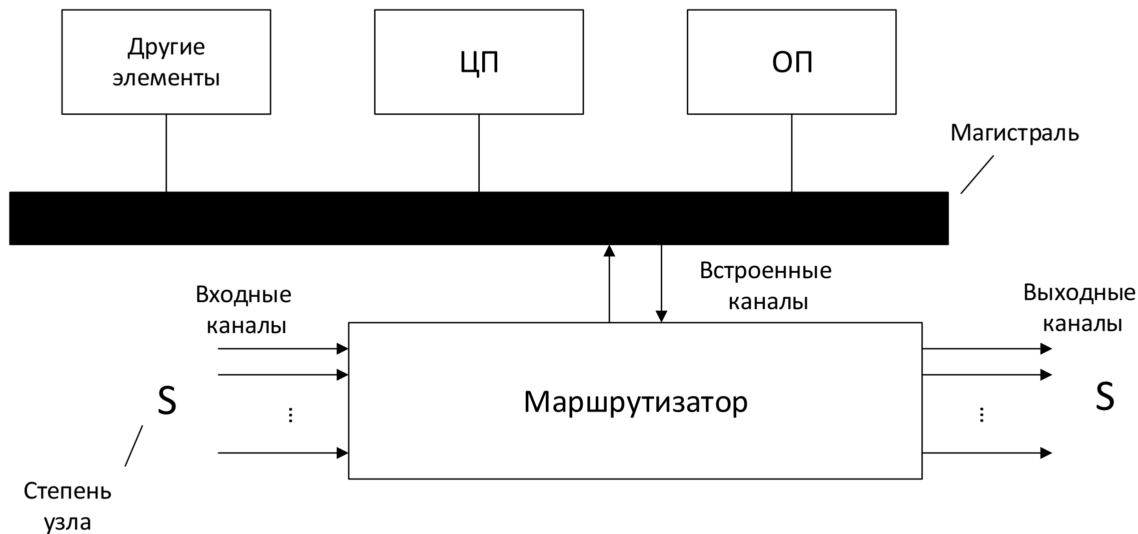
3. Пассивная избыточность (Мажоритарная логика Voter)
   рис
4. Активная избыточность
   А) дублирование + сравнение
   Б) резервирования (горячее, холодное)
5. Гибридная избыточность
   А) N-модульная избыточность с резервированием
   Б) Дублирование с резервирование
6. Информационная избыточность
   - это коды: контроль на четность
   Циклические коды
   _______ коды
   Коды Бергера
   Коды Хемминга
   Дублирование кода
   ______ контрольных сумм
   ______ кода
7. Программная избыточность
   Согласованная проверка
   Мандатная проверка
   Программное дублирование

1.3. ОСНОВНІ КОНЦЕПЦІЇ КОМУТОВАНИХ СИСТЕМ

Основи техніки комутації

1. Непосредственно связанные сети (или сети «точка-точка», или сети с фиксированной топологией).

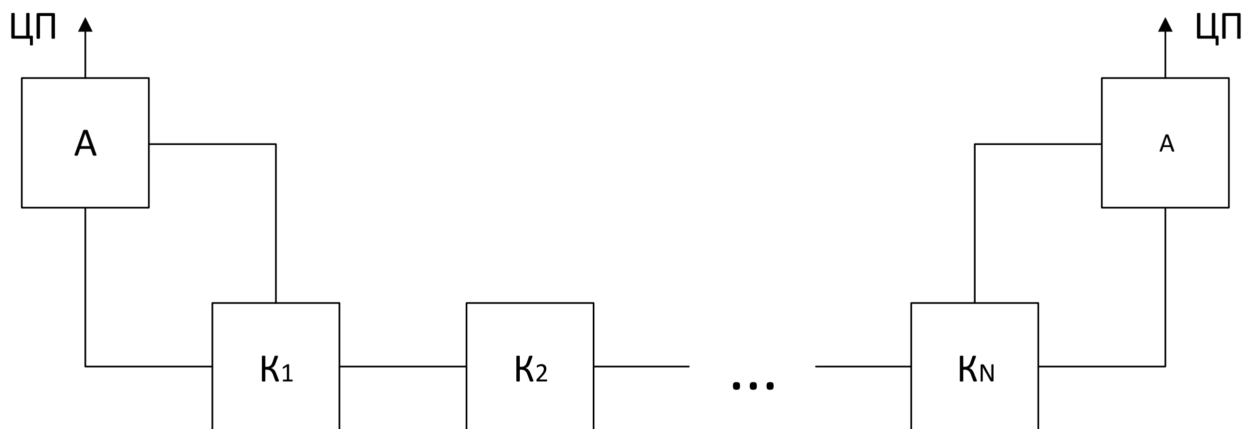
Эти сети являются весьма популярными, так как они в полной мере решают вопросы масштабируемости систем. Общим элементом в таких системах является роутер (или маршрутизатор), поэтому такие сети называют маршрутизацией.



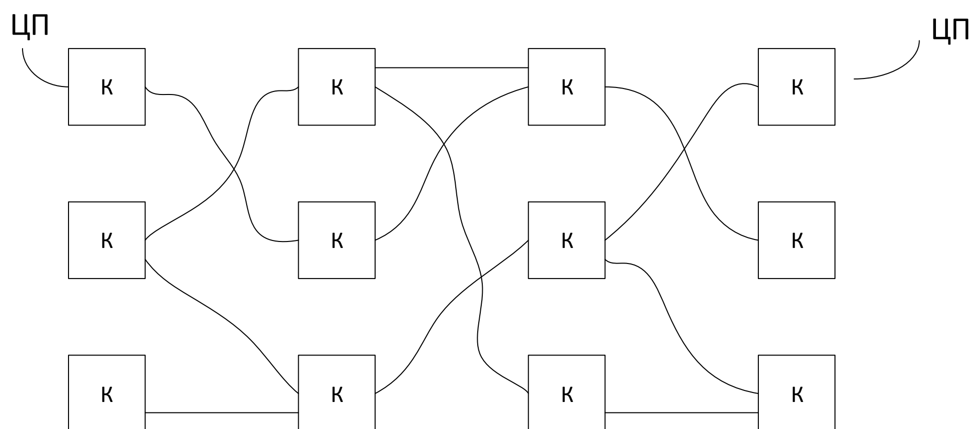
2. Коммутируемые сети или сети с реконфигурацией системы связей.

В таких сетях каждый узел должен иметь соответственный адаптер сети. С помощью этого адаптера каждый узел может подключаться к коммутатору сети. Каждый коммутатор обладает множеством портов, где каждый порт включает в себя входящие и выходящие вилки. Каждый коммутатор с помощью своих портов может подключаться к другому коммутатору, к процессору, а также может оставаться открытым (т.е. не задействованным).

В этом случае расстояние между узлами равно числу коммутаторов + 2 (количество адаптеров).

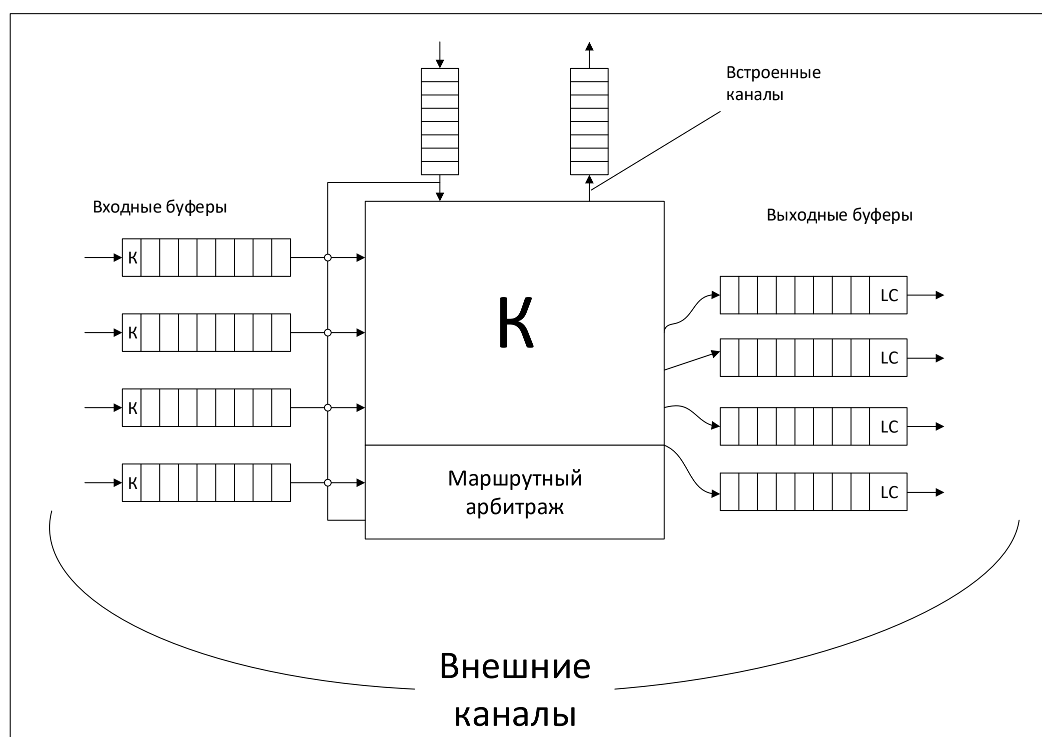


Например:



Можно рассматривать эту коммуникационную сеть как единый коммутатор.

Таким способом, мы можем сделать здесь всё, что рассматривалось в непосредственно связанных сетях.



LC – link controller (контроллер линка (связи)).

Арбитраж – это когда две входные очереди хотят связаться с i-ой выходящей.

Арбитраж решает, какая связь будет установлена, а какая подождёт.

Встроенные каналы связываются с адаптерами.

В коммутирующих сетях обычно различают 3 уровня:

1) Физический
2) Коммутационный
3) Уровень маршрутизации

На <u>физическом</u> уровне решаются вопросы управления и синхронизации передачи информации между линиями соседних маршрутизаторов.

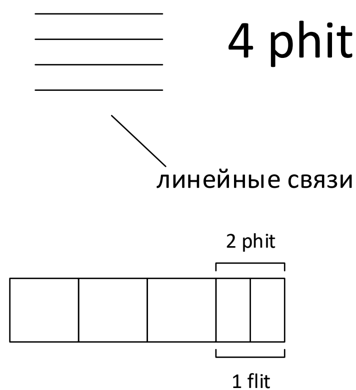
Коммутационный уровень реализует передачу информации через сеть.

На уровне маршрутизации решаются вопросы выбора выходящих буферов для каждого входящего буфера, т.е. определяется путь передачи информации.

Всё это вызывает временные и технические затраты.

Будем различать различные объёмы информации, которые являются управляемыми.

Обычно различают минимальную логическую единицу и минимальную физическую единицу. Именно минимальная логическая единица определяет минимальную единицу управления. Используется понятие «flit» для логической минимальной единицы и «phit» для физической.



Мы будем предполагать (для простоты), что 1 phit = 1 flit.

Выделим задержки:

t_r ($t_{routing}$) – время для решения вопросов маршрутизации (определения маршрута);

t_s (t_{switch}) – задержка коммутатора;

t_w – передача информации.

L – длина пакета.

W – количество битов в одном flit.

L+W – длина сообщения.

Частота коммутирующих элементов B Гц. Тогда пропускная способность: $B * W$. Задержка в канале: $\frac{1}{B}$.

Ефективність алгоритмів маршрутизації повідомлень від відправника до отримувача суттєво впливає на продуктивність комп’ютерних систем. Механізми маршрутизації визначають шлях повідомлення, при цьому потрібно враховувати стан мережі, протоколи і техніки комутації. Техніка комутації визначається розмірами одиниць інформації, тобто найменшою частиною даних, яка запитується відправником і підтверджується отримувачем. У загальному випадку кожне повідомлення може поділятися на пакети фіксованої довжини.

Пакети, в свою чергу, можуть поділятися на фліти (flits). В зв'язку з обмеженням ширини каналу для передачі одного фліту може знадобитися кілька циклів (тактів) роботи каналу. Фіт (phit) - це кількість одиниць інформації, яка може передаватися через канал за один такт. Таким чином фліт - це логічна одиниця інформації, фіт - це фізична одиниця даних, тобто число бітів, які можуть передаватися протягом одного такту. Наприклад: повідомлення складається з N пакетів, кожен пакет складається з r флітів, а кожен фліт складається з двох фітів (рис. 3.1)

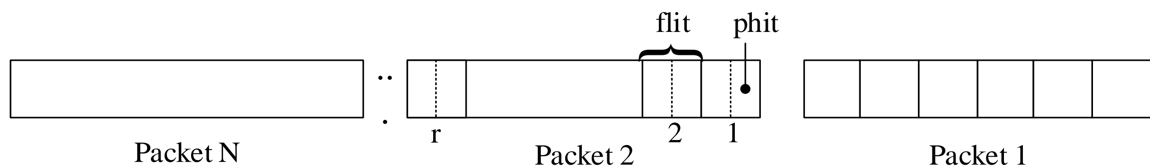


Рис. 3.1. Повідомлення, пакети, фліти, фіти

Зануримося в історичну ретроспективу відносин між розмірами пакетів, флітів і фітів в різних машинах. Так в IBM SP2 (1993) фліт дорівнює байту і розмір фліта і фіта однаковий. В Cray T3D (1993) кожен фліт складається з 8-ми 16-бітових фітів. В Cray Titan (2012) на основі Gemini Interconnect розмір фіту складає 24 біта, в Cray XC50 (2016, Rank 6, Top500, November 2019) на основі Aries interconnect розмір пакету складає вже 64 байта, а розмір фліту – 48 біт.

На рисунку 3.2 показано простий 4-фазовий асинхронний протокол пересилки з кодом підтвердженням, який дозволяє виконувати пересилання пакетів між вузлами з перевіркою отримання повідомлення. [Duato, Jose, Sudhakar Yalamanchili, and Lionel Ni. *Interconnection networks*. Morgan Kaufmann, 2003.]

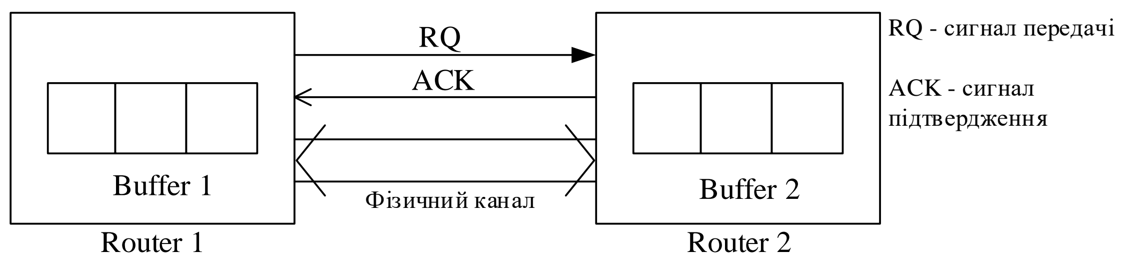


Рис. 3.2. Протокол пересилки з підтвердженням

Ефективність алгоритмів маршрутизації повідомлень від відправника до отримувача суттєво впливає на продуктивність комп’ютерних систем. Механізми маршрутизації визначають шлях повідомлення, при цьому потрібно враховувати стан мережі, протоколи і техніки комутації. Техніка комутації визначається розмірами одиниць інформації, тобто найменшою частиною даних, яка запитується відправником і підтверджується отримувачем. У загальному випадку кожне повідомлення може поділятися на пакети фіксованої довжини. Пакети, в свою чергу, можуть поділятися на фліти (flits). В зв’язку з обмеженням ширини каналу для передачі одного фліту може знадобитися кілька циклів (тактів) роботи каналу. Фіт (phit) - це кількість одиниць інформації, яка може передаватися через канал за один такт. Таким чином фліт - це логічна одиниця інформації, фіт - це фізична одиниця даних, тобто число бітів, які можуть передаватися протягом одного такту. Наприклад: повідомлення складається з N пакетів, кожен пакет складається з r флітів, а кожен фліт складається з двох фітів (рис. 3.1)

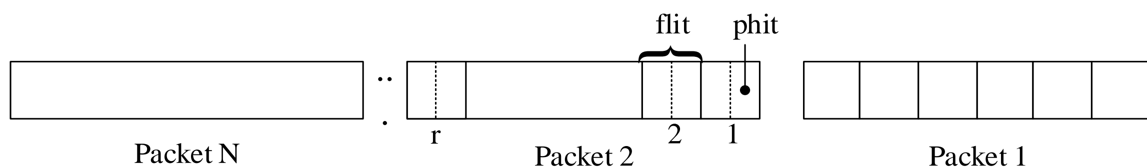


Рис. 3.1. Повідомлення, пакети, фліти, фіти

Зануримося в історичну ретроспективу відносин між розмірами пакетів, флітів і фітів в різних машинах. Так в IBM SP2 (1993) фліт дорівнює байту і розмір фліта і фіта однаковий. В Cray T3D (1993) кожен фліт складається з 8-ми 16-бітових фітів. В Cray Titan (2012) на основі Gemini Interconnect розмір фіту складає 24 біта, в Cray XC50 (2016, Rank 6, Top500, November 2019) на основі Aries interconnect розмір пакету складає вже 64 байта, а розмір фліту – 48 біт.

На рисунку 3.2 показано простий 4-фазовий асинхронний протокол пересилки з кодом підтвердженням, який дозволяє виконувати пересилання пакетів між вузлами з перевіркою отримання повідомлення. [Duato, Jose, Sudhakar Yalamanchili, and Lionel Ni. *Interconnection networks*. Morgan Kaufmann, 2003.]

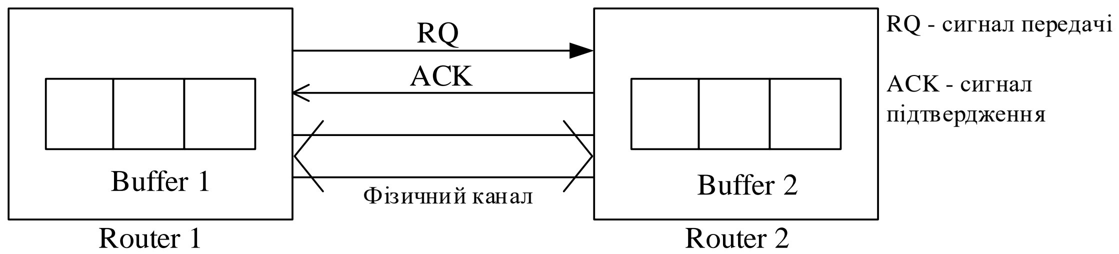


Рис. 3.2. Протокол пересилки з підтвердженням

З метою порівняння різних методів комутації будемо розглядати L-бітові повідомлення, де розмір фліта W-бітів дорівнює розміру фіта. Розмір заголовка дорівнює одному фліту. Таким чином, розмір повідомлення дорівнює $(L + W)$ бітів. Маршрутизатор вирішує питання маршрутизації на протязі t_r секунд. Частотні можливості каналу між двома маршрутизаторами дорівнюють B Гц, тобто пропускна здатність каналу дорівнює $B \cdot W$ біт в секунду, а затримка поширення через канал дорівнює $t_w = \frac{1}{B}$. Вибір одного з маршрутів в маршрутизаторі, тобто внутрішня затримка маршрутизатора (затримка комутації) дорівнює t_S. Таким чином, протягом часу t_S, W-бітовий фліт може бути переданий з входу маршрутизатора на його вихід. Припускається, що відстань між джерелом і приймачем дорівнює D лінкам.

Комутація каналів

Комутація каналів вимагає резервування шляху від відправника до отримувача ще до початку передачі даних. Це досягається за рахунок передачі заголовка, який включає в себе адресу призначення і деяку управляючу інформацію. Він просувається до вузла призначення, резервуючи при цьому фізичні канали. Коли заголовок досягає вузла призначення і підтверджує прибуття, то це означає встановлення повного маршруту. Даний шлях звільнюється після проходження останніх бітів повідомлення. Після встановлення маршруту повідомлення може передаватися з використанням всієї ширини каналу. В кінці передачі виконується підтвердження прибуття.

Комутація каналів ефективна при передачі довгих повідомлень, тобто коли час передачі є суттєво більшим ніж час з'єднання. Недоліком комутації каналів є

резервування фізичних шляхів на весь період передачі повідомлення, і відповідно, можливі блокування інших повідомлень.

Час передачі повідомлення при комутації каналів складається з часу встановлення шляху і часу передачі даних. Зазначимо, що маршрутна інформація буферизується в кожному маршрутизаторі, а дані не буферизуються, тому що канал виступає в ролі дротового зв'язку між відправником і отримувачем. Цей зв'язок може використовуватися в асинхронному і синхронному режимах. Час передачі флітів визначається тактовою частотою для синхронного режиму та сигналами підтвердження при асинхронному режимі. Період сигналів підтвердження або синхронізації повинен бути більше ніж час розповсюдження сигналів в каналі. Це накладає практичні обмеження на швидкість способу комутації каналів в залежності від розмірів системи.

Рис. 3.3. Просторова-часова діаграма передачі даних на основі комутації каналів

На рисунку 3.3 зображено часова діаграма передачі даних від відправника до отримувача через два проміжних вузли, тобто повідомлення повинно пройти три лінки. Загальний час передачі складається з часу встановлення маршруту t_{setup} і часу передачі даних t_{data}. Час встановлення маршруту теж складається з двох частин. Шлях заголовка до отримувача включає час маршрутизації t_r, час комутації t_s і час передачі заголовка t_w. Шлях заголовка назад до відправника, який є підтвердженням маршруту, складається лише з часу комутації t_s і часу передачі заголовка t_w. Таким чином, враховуючи, що кількість лінків на шляху

дорівнює D, розмір повідомлення L, а ширина каналу B отримаємо наступну формулу:

$$t_{transfer} = D\big(t_s + 2(t_s + t_w)\big) + t_w \left\lceil \frac{L}{W} \right\rceil$$

В даній формулі припускається, що зворотній шлях заголовку не потребує маршрутизації (тому t_r не подвоюється), та що передача даних починається миттєво після повернення заголовку. Розглянемо наступну техніку комутації, а саме, комутацію пакетів.

Комутація пакетів

В способі маршрутизації на основі комутації пакетів повідомлення розділяються на пакети, які мають фіксовану довжину. При цьому перші декілька байтів пакета розглядаються як його заголовок та містять маршрутну і управляючу інформацію. Кожен пакет індивідуально передається від відправника до отримувача. Пакети повністю буферизуються в кожному проміжному вузлі і потім передаються наступному вузлу, тому такий спосіб передачі іноді називають store-and-forward комутація. В заголовку міститься інформація про наступний маршрутизатор, що дозволяє визначити вихідний лінк.

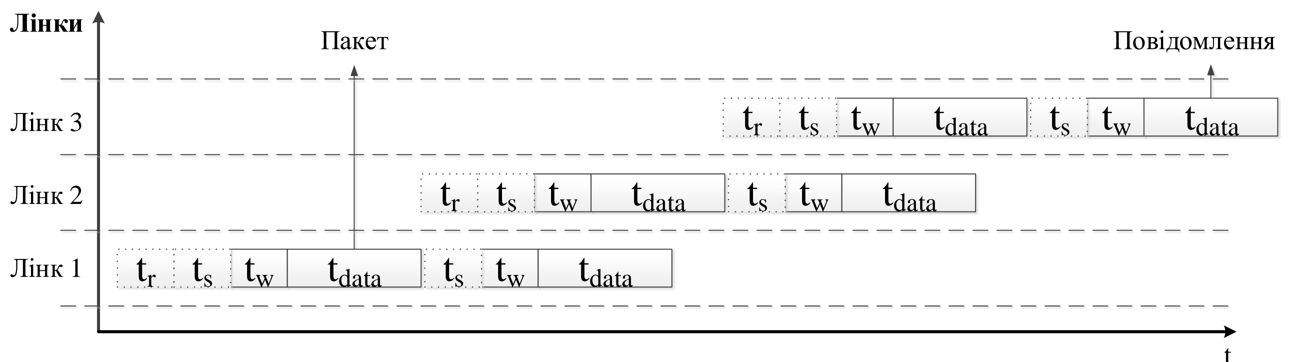


Рис. 3.4. Просторова-часова діаграма передачі даних на основі комутації пакетів

Як видно з рисунку 3.4., повідомлення розподіляється на пакети і виконується маршрутизація. Таким чином загальний час передачі дорівнює:

$$t_{packet} = D\ (\ t_r + (t_w + t_s)\ \left\lceil \frac{L}{W} + 1 \right\rceil)$$

Даний спосіб комутації є більш ефективним для коротких повідомлень. На відміну від комутації каналів, пакети не блокують маршрут для інших пакетів. Однак, це збільшує витрати, тому що повідомлення потрібно розділити на частини, і потім виконувати окрему маршрутизацію для цих частин. Також, для того, щоб зібрати повідомлення, після отримання всіх пакетів, отримувачу необхідно знати їх порядок.

Віртуальна конвеєрна комутація пакетів

Спосіб комутації пакетів передбачає відправку наступного пакету після того, як попередній пакет повністю відправлено. Але можливо значно прискорити передачу даних, якщо припустити що маршрутизатор має декілька вільних буферів, які може використати для відправки нових пакетів. Таким чином, можна організувати конвеєрну віртуальну комутацію пакетів (рис 3.5.).

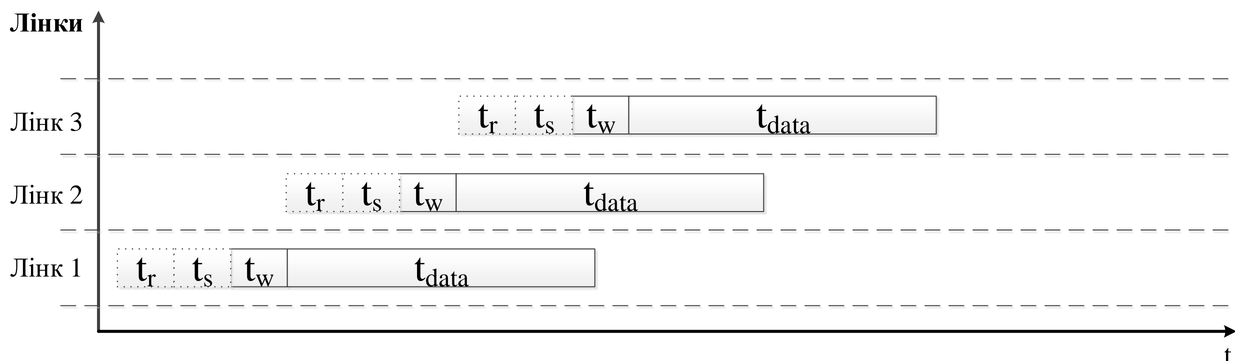


Рис. 3.5. Просторова-часова діаграма для віртуальної конвеєрної комутації

Як видно з рисунку 3.5. передача наступних пакетів починається зразу після прокладання маршруту, але при наявності вільних буферів. Такий спосіб називається virtual cut-through маршрутизацією.

$$t_{VCT} = D\ (t_r + t_s + t_w) + \max(t_s, t_w) \left\lceil \frac{L}{W} \right\rceil$$

Загальний час передачі повідомлення значно скорочується ніж при комутації пакетів за рахунок конвеєрної передачі даних.

Черв'ячна комутація пакетів

Спосіб віртуальної конвєрної комутації пакетів передбачає відправку наступного пакету зразу після прокладання маршруту, але при наявності вільних буферів. В зв'язку з цим буфери мають великий розмір, і відповідно збільшують вартість маршрутизаторів. Черв'ячна комутація пакетів використовує буфери значно менших розмірів. Кожен пакет розподіляється на фліти і, відповідно, управління передачею даних здійснюється на рівні флітів. Зазвичай повідомлення є достатньо великим, тому займає декілька буферів на різних маршрутизаторах. На рис. 3.6. показано черв'ячну комутацію пакетів.

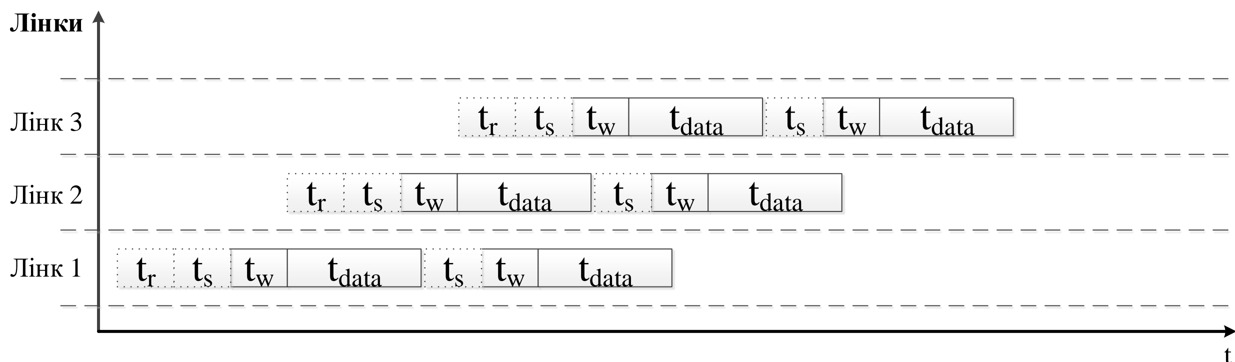


Рис. 3.6. Просторова-часова діаграма для віртуальної конвеєрної комутації

Як видно з рисунку 3.6. передача наступних флітів починається зразу після прокладання маршруту, але при наявності вільних буферів. Такий спосіб називається wormhole маршрутизацією.

$$T_{wormhole} = D(t_r + t_s + t_w) + max\ (t_s, t_w)\ \left\lceil \frac{L}{W} \right\rceil$$

Таким чином, розмір буферу - це головна різниця між конвеєрною та черв'ячною комутаціями. В конвеєрній комутації розмір буферу дорівнює розміру пакету, а в черв'ячній - розміру фліту. Зменшений розмір буферу знижує вартість, але, через велику кількість маршрутизаторів, можливі блокування.

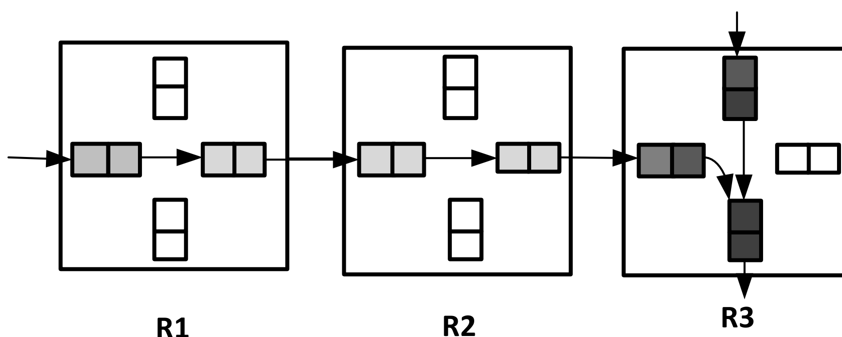


Рис.XXX. Приклад блокування для черв'ячної передачі повідомлень.

Комутація «божевільного поштаря»

Підвищення ефективності комутації відбувалось за рахунок конвеєризації передачі цілих пакетів (конвеєрна комутація) та зменшення пакету до фліту (черв'ячна комутація). Наступним кроком збільшення рівня конвеєризації є спроба зменшити можливі затримки за рахунок використання особливостей топологій. Розглянемо цей спосіб на основі механізму божевільного поштаря, для топології решітка (рис 3.7.).

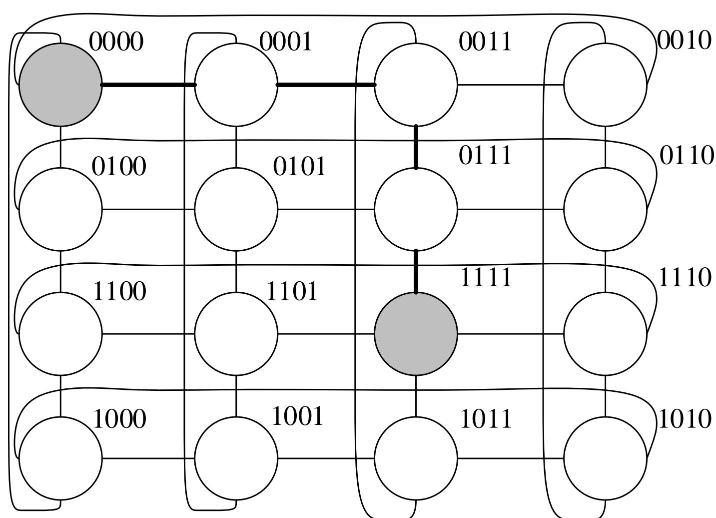


Рис. 3.7. Комутація «божевільного поштаря» в меш-топології

Розглянемо 2-D меш мережу з пакетами, які мають двохфлітові заголовки. Тобто, перший фліт відповідає за маршрутизацію по горизонталі, а другий фліт - по вертикалі. Наприклад, потрібно передати повідомлення від вузла 00.00 до вузла 11.11. В розглянутих раніше способах комунікації дані починають передаватися лише після формування обох флітів. Спосіб божевільного поштаря

дозволяє розпочати передачу вже після формування хоча б одного фліту. Для нашого прикладу, якщо визначено другий фліт ($11.\mathbf{11}$), то вже відразу починається передача даних: $00.00 \rightarrow 00.01 \rightarrow 00.11$. В цей момент відбувається уточнення першого фліту отримувача ($\mathbf{11}.11$), і завершується передача даних: $00.11 \rightarrow 01.11 \rightarrow 11.11$.

$$t_{madpostman} = (t_s + t_w)D + \max(t_s, t_w)\,W + \max(t_s, t_w)\,L$$

Отже, божевільний поштар дозволяє передачу не чекаючи повної адреси.

Віртуальні канали

Розглянуті вище механізми комунікації вимагають монопольного доступу до фізичного каналу. Тобто, якщо повідомлення займає буфер каналу, то жодне інше повідомлення не має доступу до фізичного каналу. Для вирішення цієї проблеми запропоновано розподіляти фізичні канали на декілька логічних або віртуальних каналів. При цьому кожен однонаправлений віртуальний канал реалізується на основі пари буферів (Рис 3.8.).

Рис. 3.8. Віртуальні канали

Віртуальні канали вирішують задачу блокування в черв'ячній комунікації за рахунок поділення фізичного каналу на логічні, які мають пропорційно меншу пропускну здатність. Також віртуальні канали використовуються для зменшення затримок передачі повідомлень, і збільшення пропускної здатності мережі за рахунок виключення блокувань (рис 3.9.).

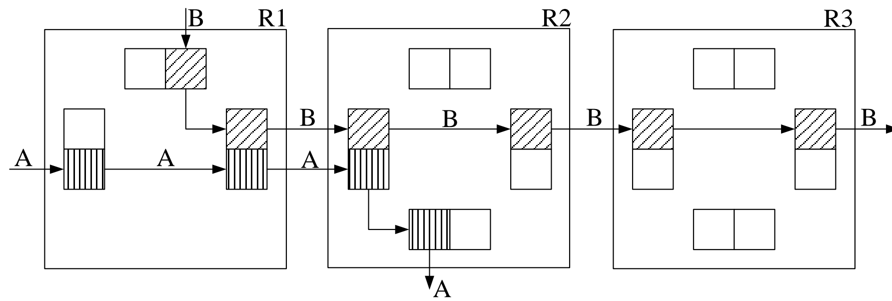


Рис. 3.9. Використання віртуального каналу для виключення блокувань

На рис. 3.9. показано два повідомлення A і B, яким потрібно перейти між маршрутизаторами R_1 і R_2. При відсутності віртуальних каналів повідомлення A буде блокувати повідомлення B до кінця передачі. Але, додаткові буфери дозволяють створити логічні канали та уникнути блокувань. Особливо ефективний цей спосіб, якщо повідомлення A набагато довше ніж B. При цьому, деякий час обидва повідомлення будуть передаватися одночасно, а після закінчення передачі повідомлення B,уся пропускна здатність каналу буде належати повідомленню A.

Гібридні способи комунікації

Під гібридними способами комунікації будемо розуміти об’єднання особливостей декількох, зазвичай двох, способів комунікації. Гібридні способи можуть показувати більшу ефективність в окремих спеціалізованих топологіях. Розглянемо два таких способи.

Кешована черв’ячна комутація

Даний спосіб комутації об’єднує особливості черв’ячної комутації з комутацією пакетів. Формат пакетів і повідомлень вибирається з урахуванням топології комунікаційних мереж. Розглянемо приклад такої багатокальної мережі, що заснована на $omega$-топології, яка використовує комутацію 4×4 з дуплексними лінками. В даному прикладі, система складається з модулів, а кожен модуль складається з 16 процесорів, та двухканального комутатора із 8-ю комутуючими елементами. Такий модуль називається - фрейм (Рис 3.10.).

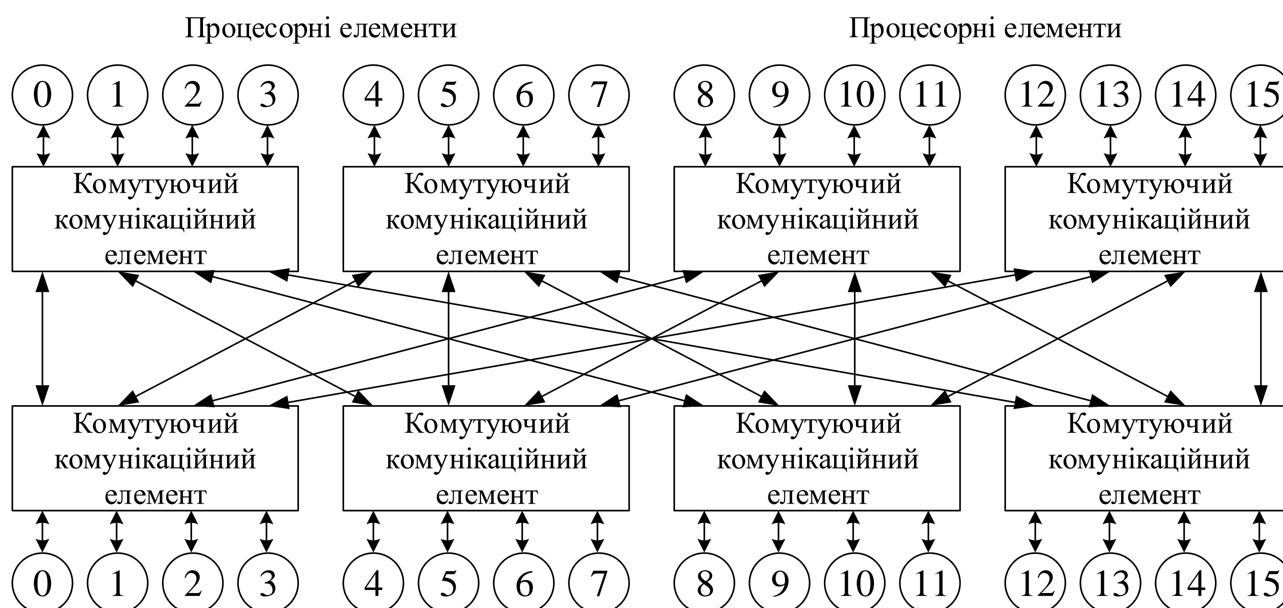


Рис 3.10. Фрейм з 16 процесорів і 8 комутуючих елементів

Довжина пакетів - до 256 флітів, розмір фліту - 1 байт, і дорівнює ширині каналу. Перший фліт повідомлення містить довжину повідомлення, наступний - маршрут. Для кожного фрейма, достатньо одного фліту для маршрутизації. Розглянемо склад фліту для маршрутизації (рис 3.11.): перші три біти - це адреса порта другого комутатору, потім розділяючий біт, наступні три біти - номер порту першого комутатору, сьомий - управляючий біт.

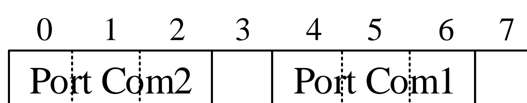


Рис 3.11. Фрейм з 16 процесорів і 8 комутуючих елементів

Повідомлення маршрутизуються за способом черв’ячної передачі, до того часу поки відсутні блокування. Якщо повідомлення блокується то створюється спеціальний 64 бітний канал, для пересилання заблокованих частин (порцій) в кеш пам’ять. Після розблокування ці частини виштовхуються з кеш пам’яті в вихідний порт.

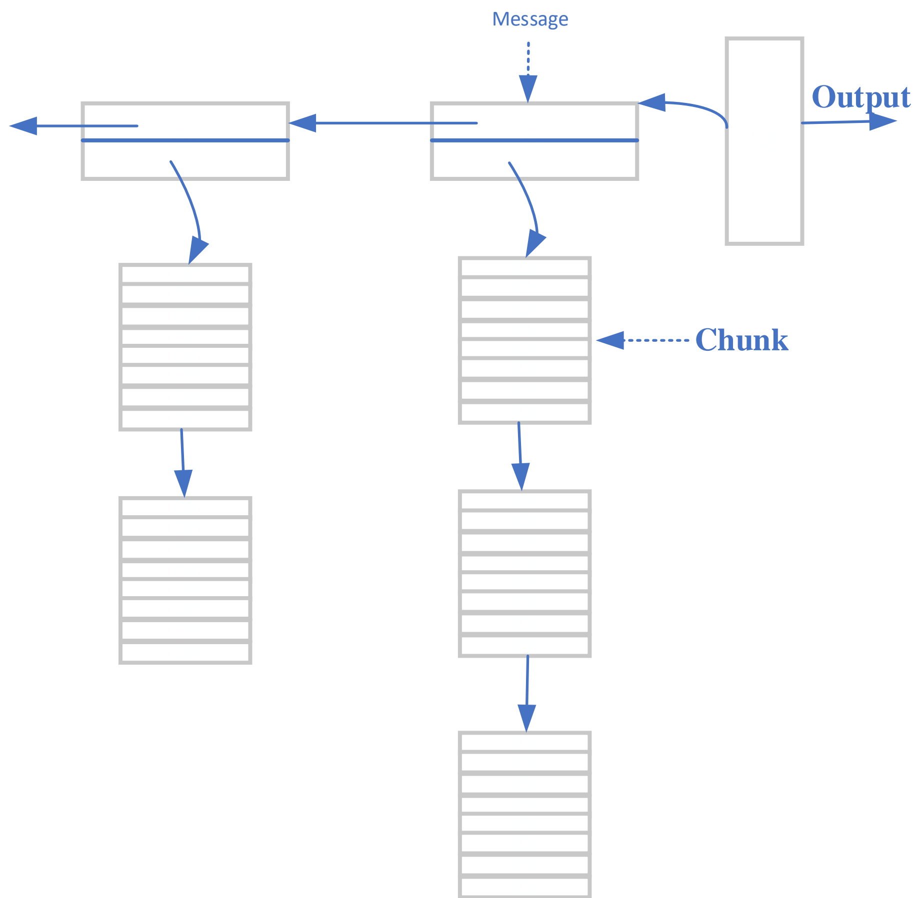


Рис.ХХХ, Логічна організація буферів повідомлень, представлених флітами.

c

v

Конвеєрна комутація каналів

Конвеєрна комутація каналів базується на способі комутації каналів та способі черв'ячної маршрутизації. Але, на відміну від комутації каналів в даному випадку використовуються віртуальні канали. Та, на відміну черв'ячної передачі, заголовок прокладає повний маршрут, і відповідно, дані не йдуть одразу за заголовком. За рахунок цього, в разі відмови, заголовок може бути повернений в попередній маршрутизатор і шукати альтернативний маршрут (рис. 3.12.).

Рис. 3.12. Просторова-часова діаграма передачі даних на основі конвеєрної комутації каналів

Після знаходження маршруту до одержувача, заголовок повертається назад до відправника, і цим підтверджує наявність маршруту. Потім, фліти даних передаються до одержувача способом черв'ячної комутації.

Цей спосіб дозволяє заголовку виконувати повернення до попереднього маршрутизатора при неможливості подальшого шляху через відмову. Тим самим, шукаються і резервуються віртуальні канали, що дозволяє створити маршрут, який обходить всі відмови.

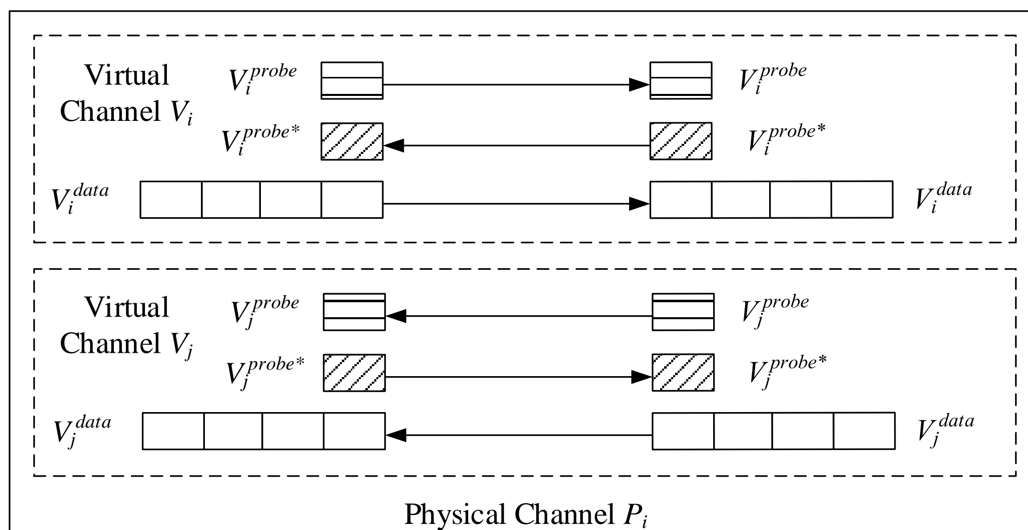


Рис. 3.13. Віртуальні канали для конвеєрної комутації каналів

На рис. 3.13. зображено фізичний канал P_i, в якому організовано два віртуальні канали V_i і V_j. Спочатку по каналу передається заголовок V_i^{probe}. Якщо заголовок зміг прокласти маршрут, то назад відправляється підтвердження V_i^{probe*}. Після цього починається передача даних V_i^{data}. Таким чином, кожен

віртуальний канал здійснює передачі 3-х типів: заголовка, підтвердження та даних.

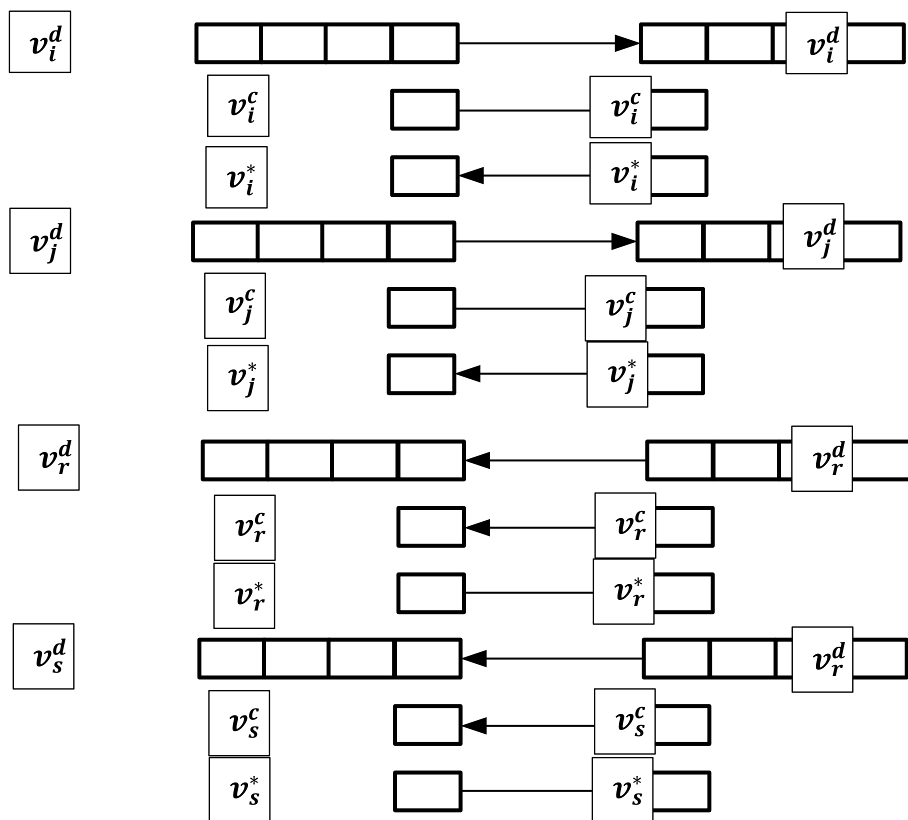


Рис.XXX. Віртуальна модель конвеєрної комутації каналів.

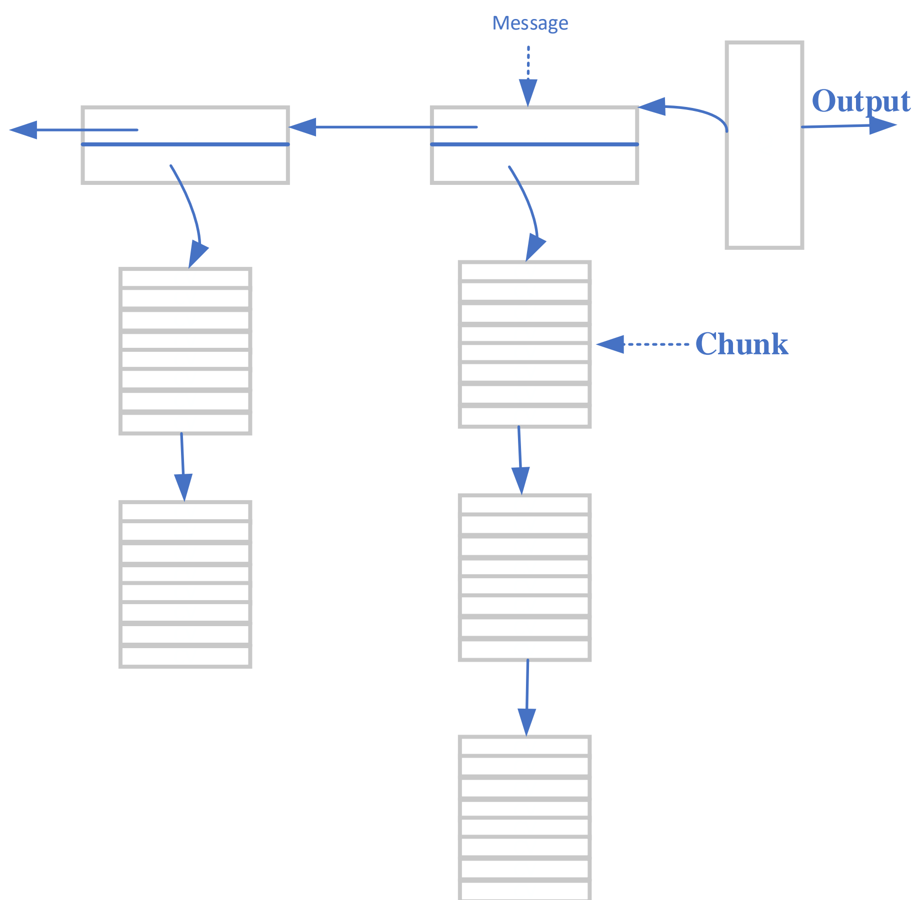


Рис.XXX, Логічна організація буферів повідомлень, представлених флітами.

с

Scouting Switching – Коммутация с упреждающей разведкой.

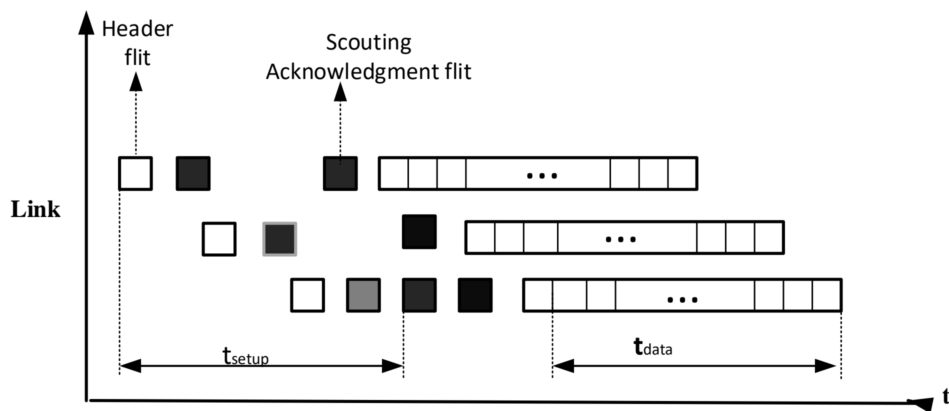


Рис.XXX. Просторово-тимчасова діаграма передачі повідомлень при суміщенні в часі комутації каналів і червячної передачі

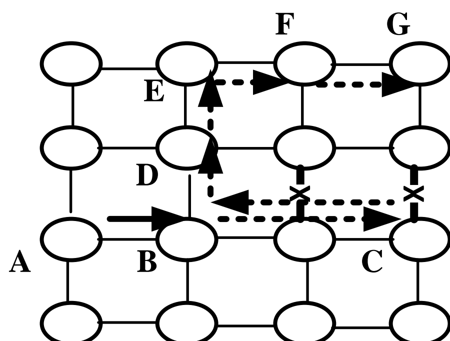


Рис.XXX. Приклад відмовостійкої маршрутизації у разі комутації з розвідкою в авангарді.

Myrinet

Мережеву технологію Myrinet представляє компанія Myricom,, яка вперше запропонувала свою комутаційну технологію в 1994 році. Таким чином, технологія Myrinet заснована на використанні багатопортових комутаторів при мережевих обмеження на довжину зв'язків між вузлами і картами комутаторів. Вузли з'єднуються один з одним через мережу комутаторів (до 128 портів) [ ]. На рис.XXX наведено приклад з’єднань у мережі Myrinet.

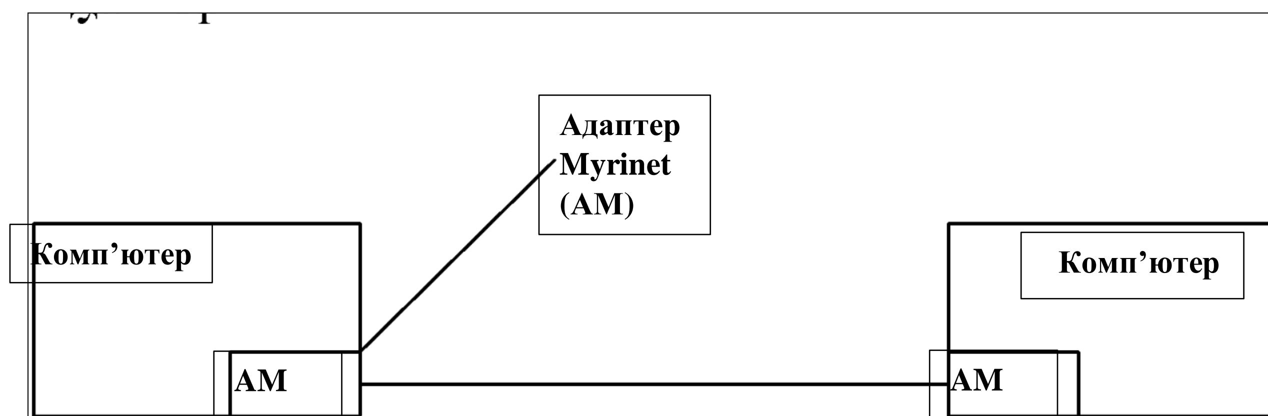


Рис. 1. Пряме з’єднання лінків адаптерів.

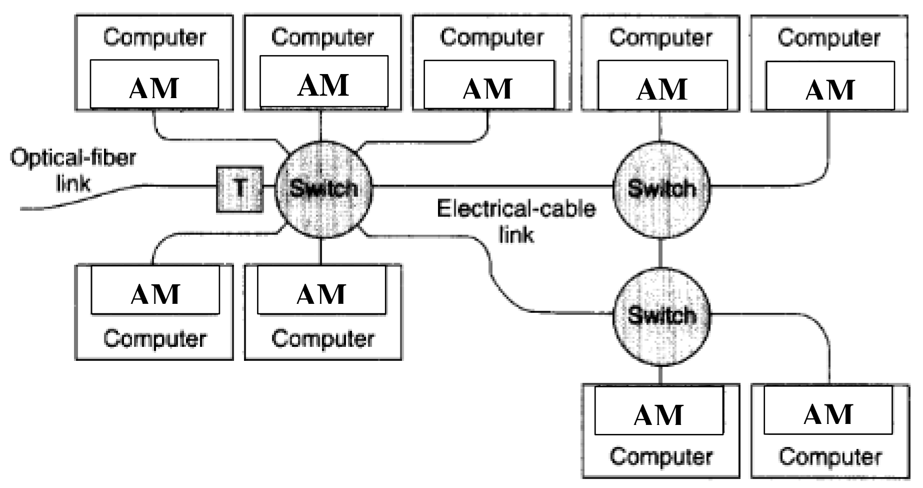


Рис. 2. Комп’ютерні системи, побудовані з використанням адаптерів і комутаторів Myrinet.

Myrinet за структурою аналогічний сегментам Ethernet, з'єднаних за допомогою комутаторів. При цьому Myrinet може передавати одночасно кілька пакетів зі швидкістю, близькою до 2Гбіт / с. На відміну від некомутованих Ethernet і FDDI мереж, які поділяють загальну середу передачі, сукупна пропускна здатність Myrinet зростає зі збільшенням кількості машин. Сьогодні Myrinet часто використовують як локальну мережу, проте з урахуванням помірної вартості, високій швидкості, малим часом затримки, прямої комутації Myrinet широко використовується для об'єднання комп'ютерів в кластери.

Пакети Myrinet можуть мати будь-яку довжину, а це означає, що вони можуть включати в себе інші пакети, включаючи і IP-пакети. Об'єднання обчислювальних вузлів з адаптерами Myrinet в мережу здійснюється за допомогою комутаторів, які зараз мають 4, 8, 12 і 16 портів. У комутаторах використовується передача пакетів шляхом встановлення з’єднання на час передачі, для маршрутизації повідомлень застосовується алгоритм прокладки шляху (wormhole). Комутатори мають власне живлення і мають можливість функціонувати автономно. Конфігурація комутаторів Myrinet ілюструється на рис.XXX і рис.XXX.

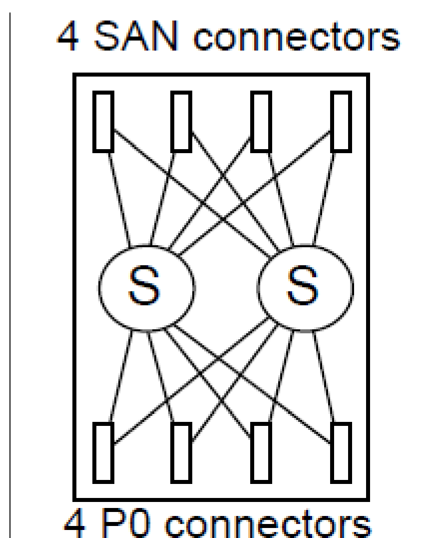


Рис.XXX. Базовий 4-портовий комутатор Myrinet

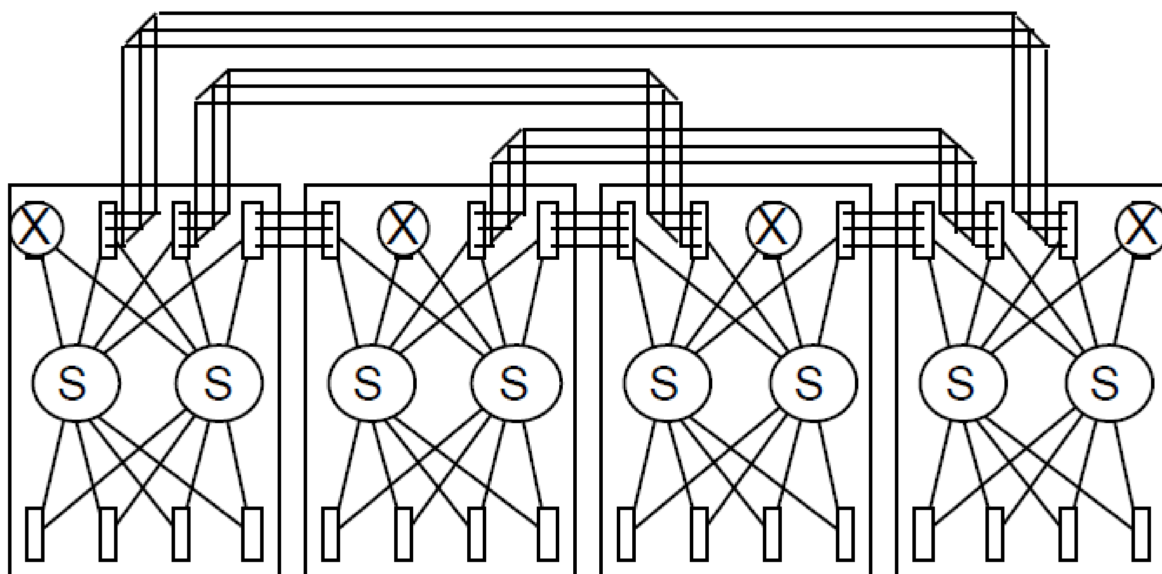


Рис.XXX. Складовий 16-портовий комутатор Myrinet

На рис.XXX ілюструється конструктивна організація комутаторів Myrinet.

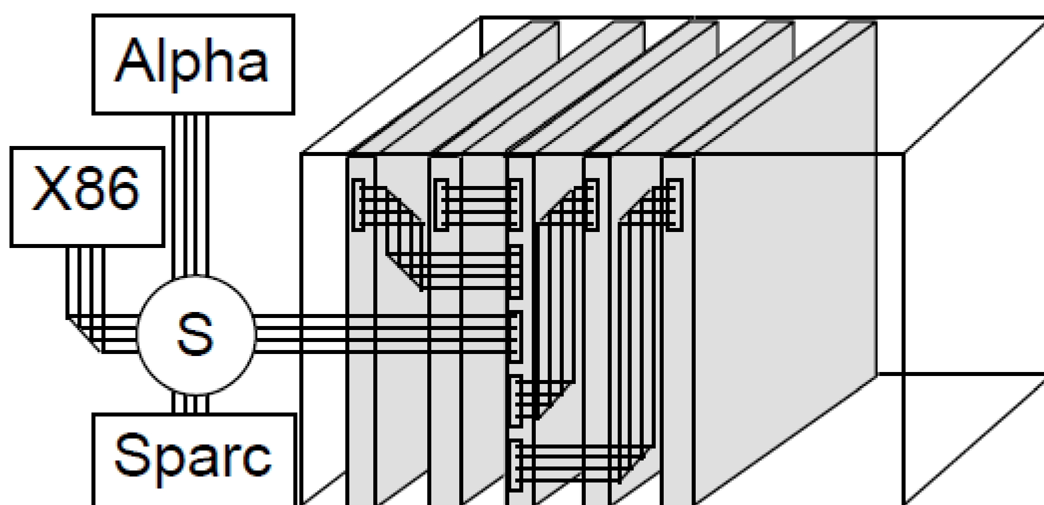
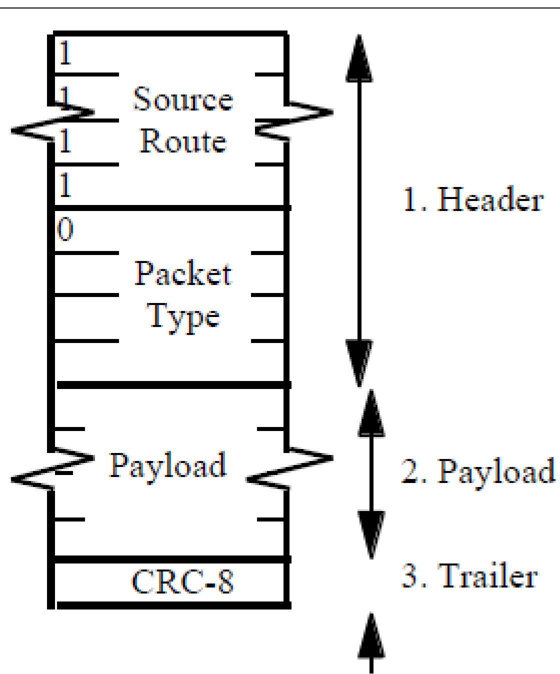


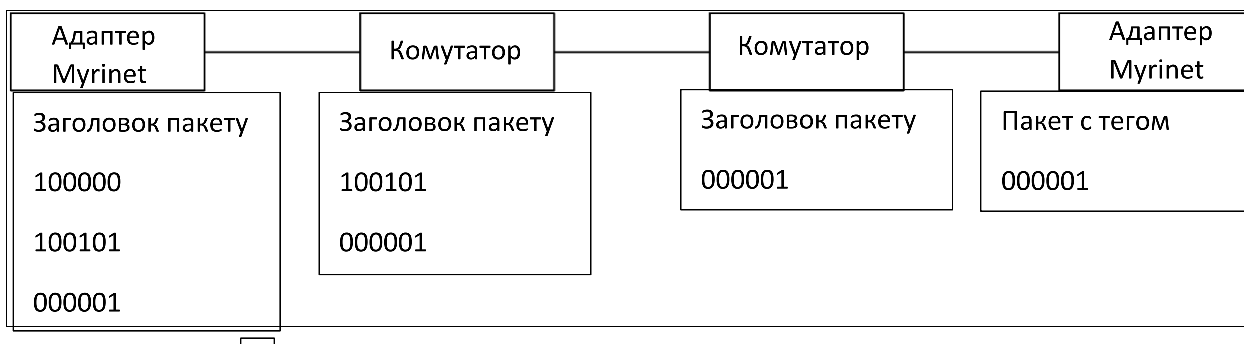
Рис.XXX. Конструктивна організація 16-портового комутатора Myrinet

Для построения больших сетей используются технологии Fat Tree, сети Клоза

Пакети Myrinet можуть мати будь-яку довжину, а це означає, що вони можуть включати в себе інші пакети, включаючи і IP-пакети. _Структура пакету представлена на рис.XXX, з якого випливає, що в якості метода маршрутизації в системах на основі Myrinet використовується маршрутизація в джерелі.



На фізичному рівні лінки Myrinet складаються з 9 провідників: 8 бітів - дані, що генеруються за станом 9-го біта як байт даних або керуючий символ. При цьому на кожному лінку забезпечується управління потоком і контроль помилок.



Скорость передачи – 3 варианта
Номинальный символьный период
Для Myrinet 640 – 640 Mbit/sec
Для Myrinet 1280 – 1280 Mbit/sec
Для Myrinet 2560 – 2560 Mbit/sec

Myrinet выгодно отличается от других сред передачи простой концепции и аппаратной реализацией протоколов. Она содержит ограниченный набор средств управления трафиком, использующих приливно-отличной буфер, управляющие символы и ____ _____

Логика реализации Myrinet протокола

ППК

Вхідна логіка протоколу

Приливно-відливний буфер

Вхідні дані

Сигнал переповнення

Керування адаптером

Вихідна логіка протоколу

Вихідна лінія

Адаптер

Stop

Go

32 bytes

16 bytes

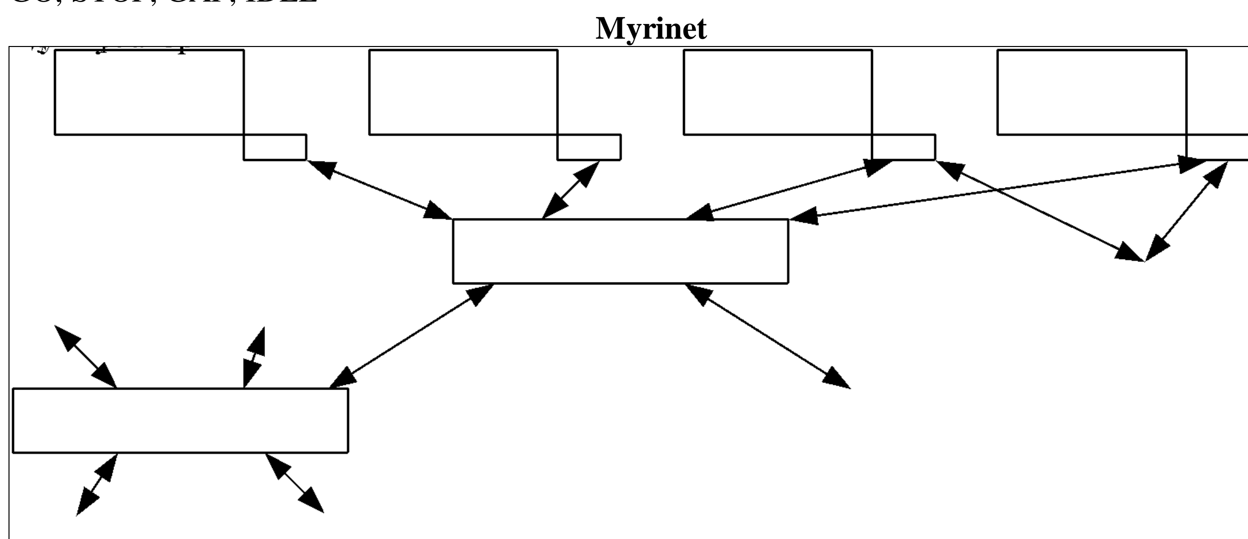
32 bytes

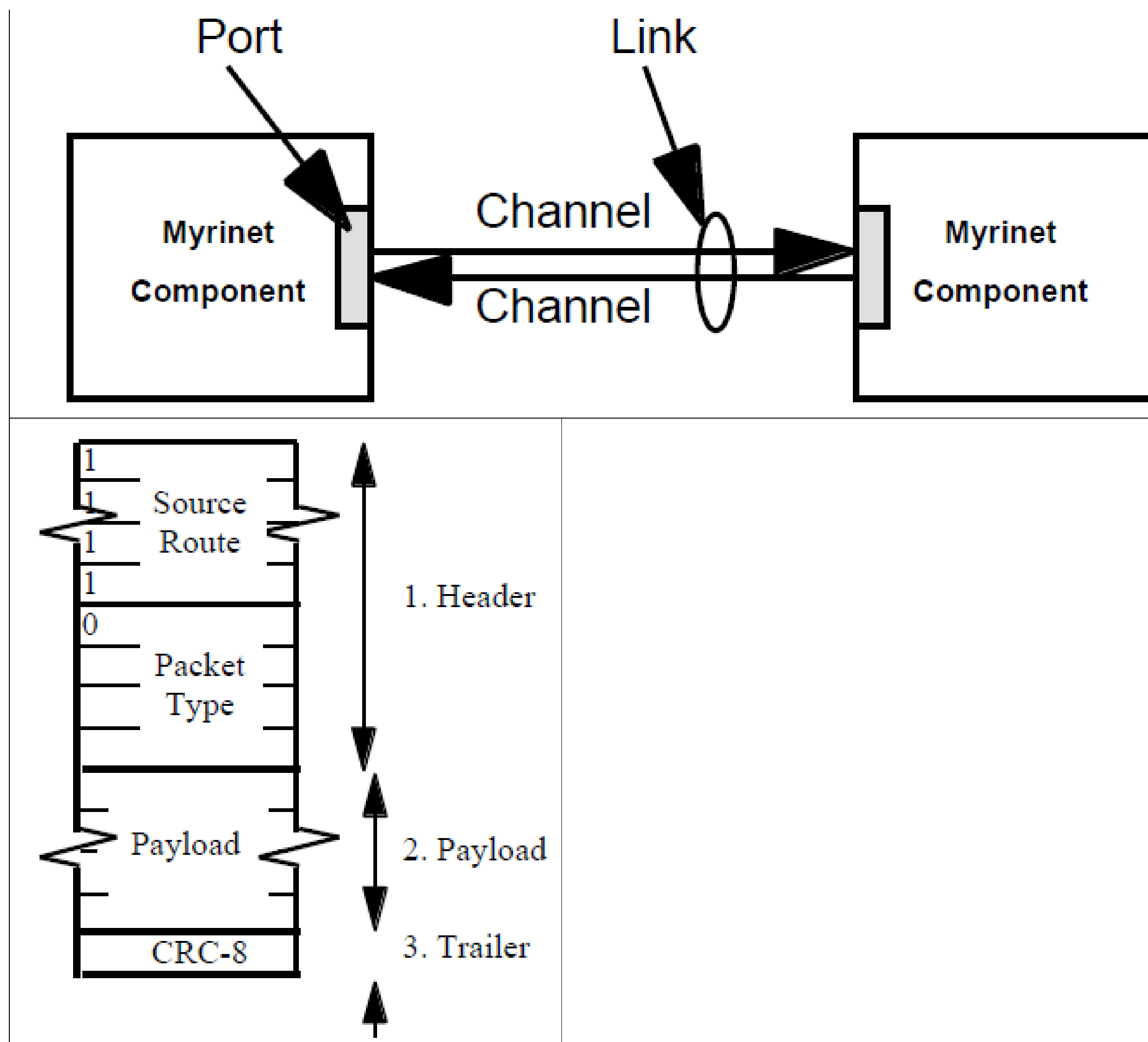
РИС

Управляемые символы

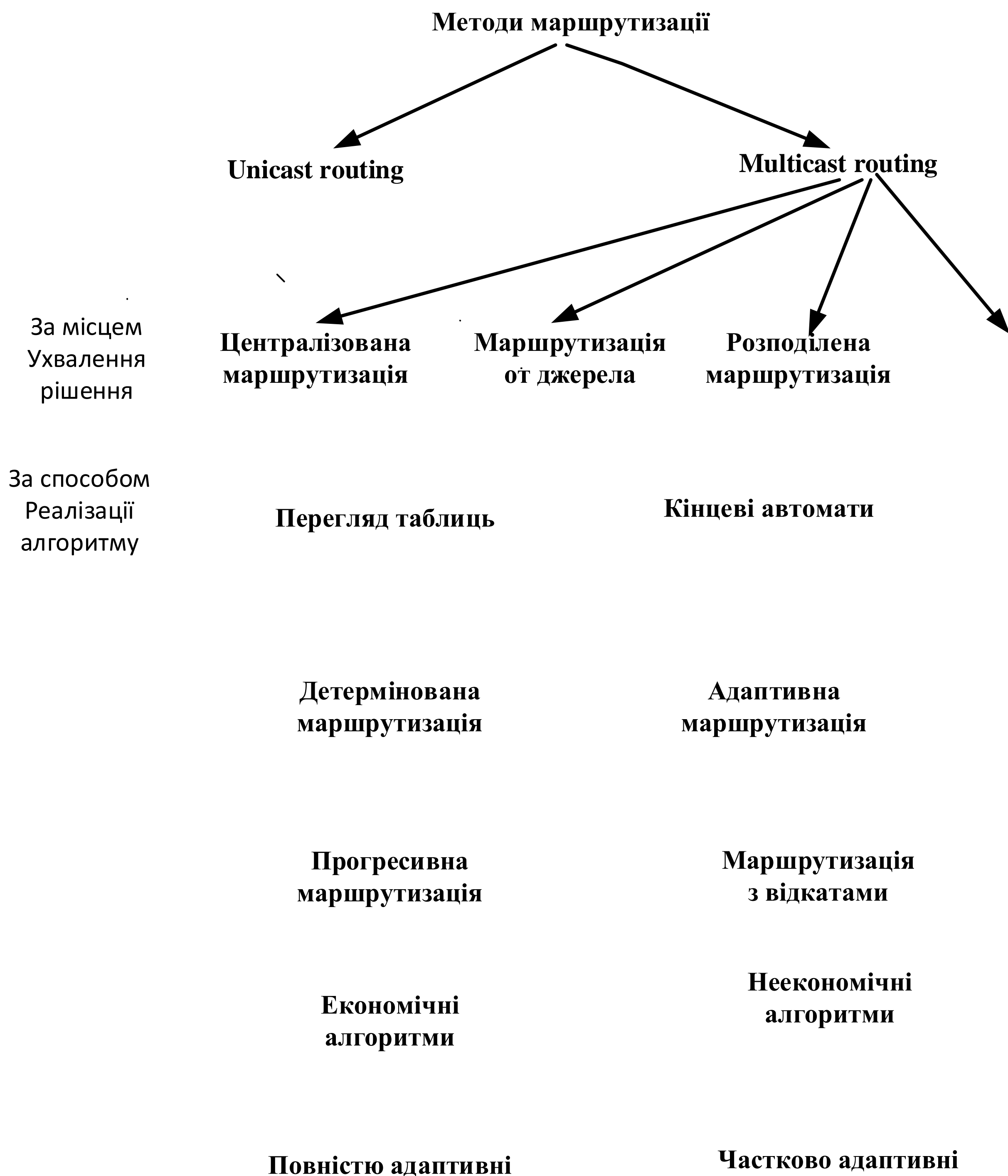
GO, STOP, GAP, IDLE

Myrinet





Port
Link
Myrinet
Component
Channel
Channel
Myrinet
Component
1
1
1
1
Source
Route
0
Packet
Type
Payload
CRC-8
1. Header
2. Payload
3. Trailer


Методи маршрутизації
Unicast routing
Multicast routing
За місцем
Ухвалення
рішення
Централізована
маршрутизація
Маршрутизація
от джерела
Розподілена
маршрутизація
За способом
Реалізації
алгоритму
Перегляд таблиць
Кінцеві автомати
Детермінована
маршрутизація
Адаптивна
маршрутизація
Прогресивна
маршрутизація
Маршрутизація
з відкатами
Економічні
алгоритми
Неекономічні
алгоритми
Повністю адаптивні
Частково адаптивні

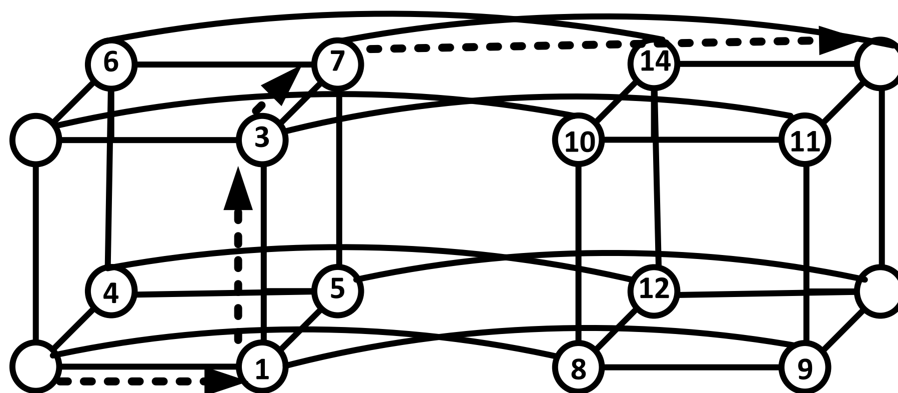


Рис.XXX. Маршрутизація у гіперкубі без відмов

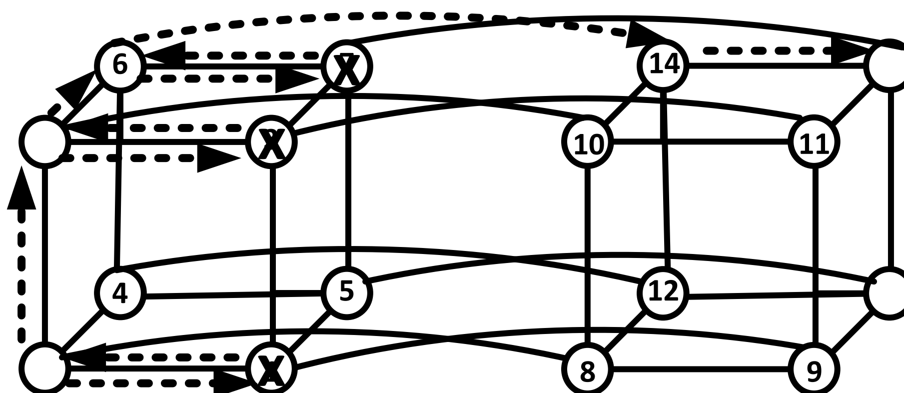


Рис.XXX. Маршрутизація у гіперкубі з відмовами.

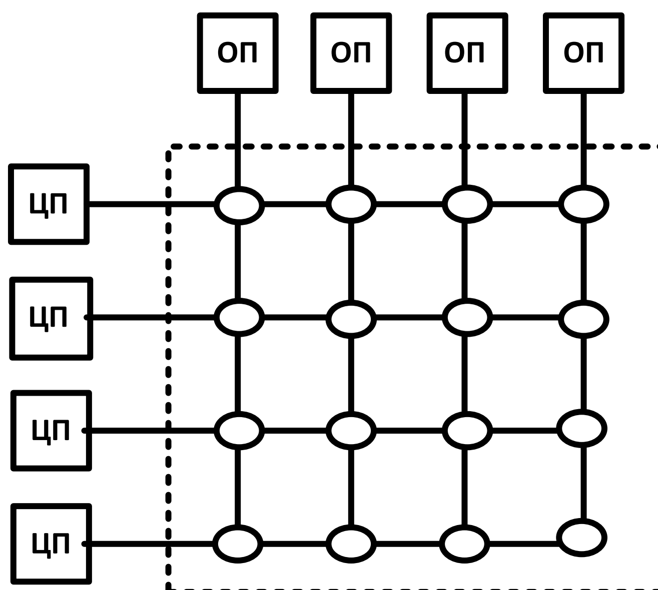


Рис.XXX. Мультипроцесорна організація з матричним комутатором.

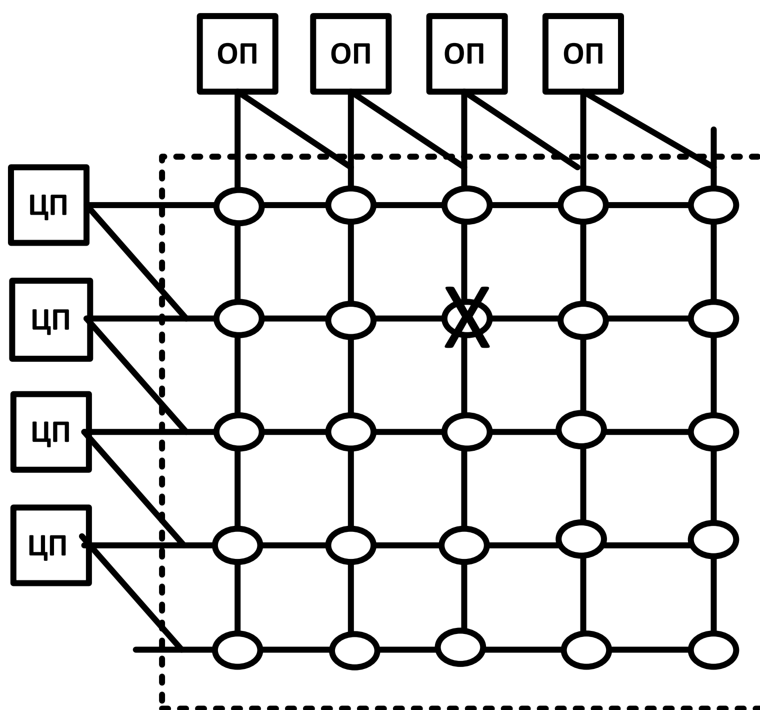


Рис.XXX. Надлишкова мультипроцесорна організація з матричним комутатором.

м

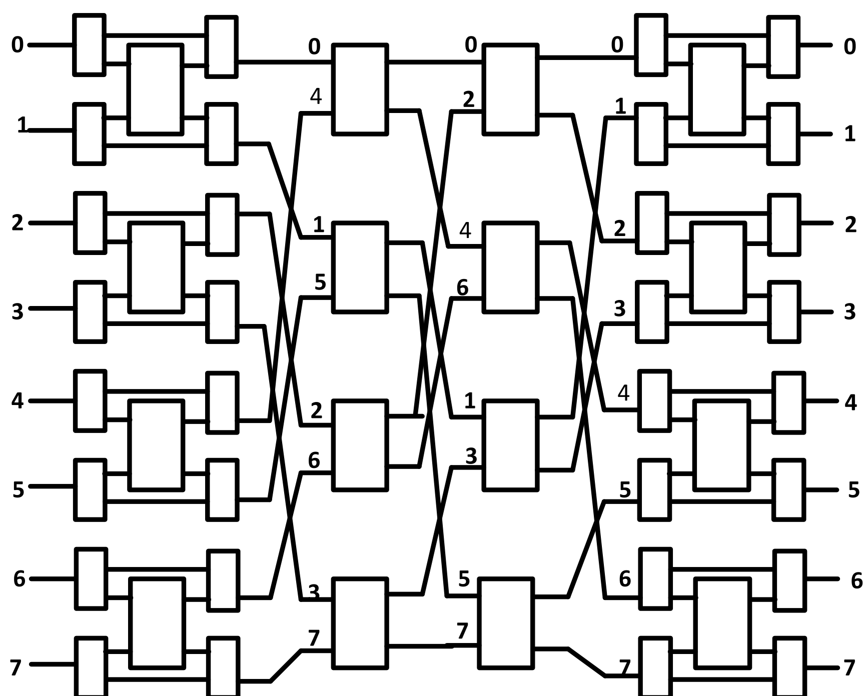


Рис.ХХХ. Мережа Омега.

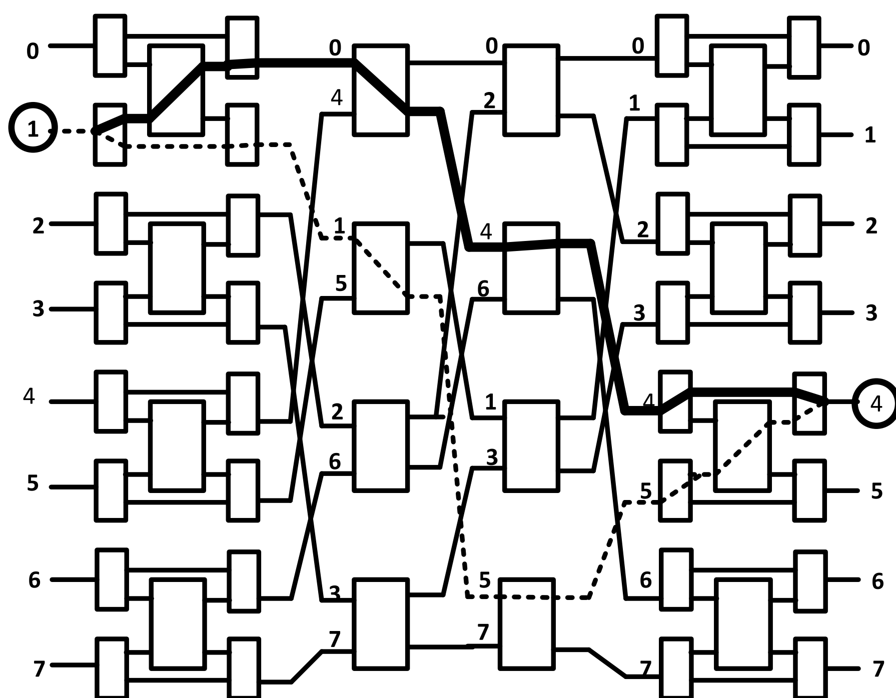


Рис.ХХХ. Надлишкова мережа Омега.

м

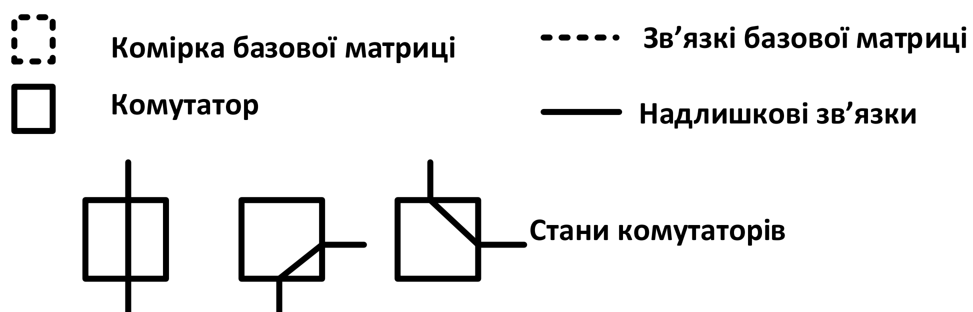


Рис.ХХХ. Меш-топологія, яка реконфігурується.

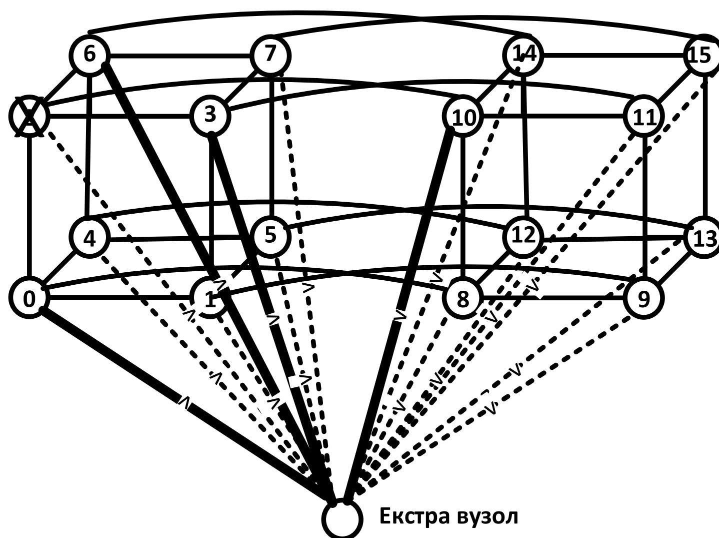


Рис.XXX. Спосіб відновлення вихідного стану гіперкуба на основі екстра вузла.

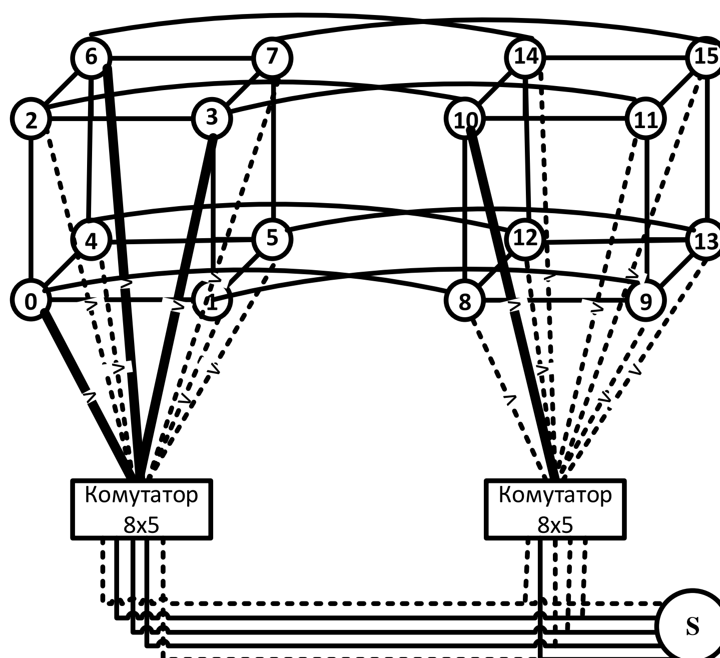


Рис.XXX. Реалізація способу відновлення вихідного стану гіперкуба на основі екстра вузла.

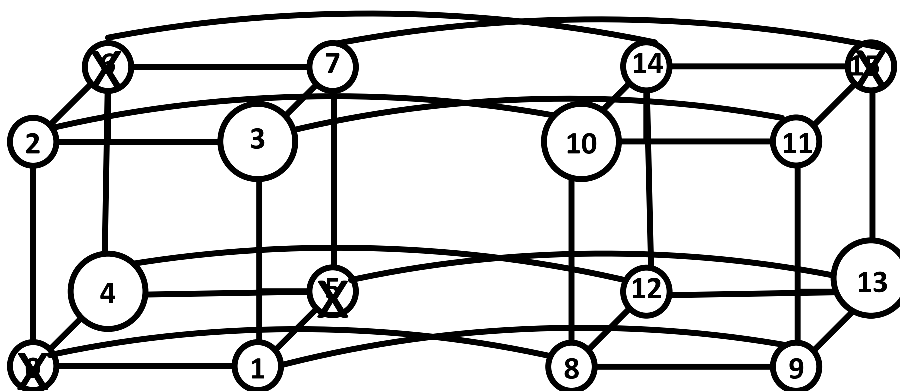


Рис.ХХХ. Гиперкуб з окремими елементами, які наділені надмірностю, і має дефектні елементи.

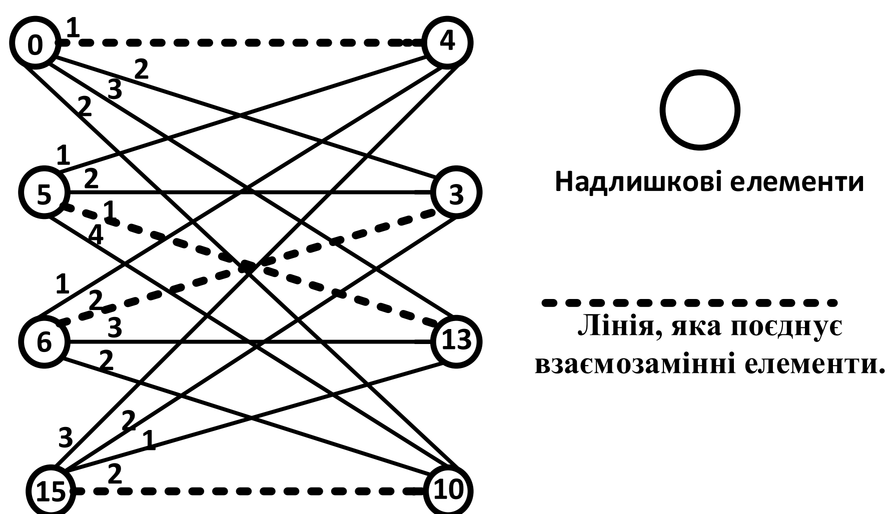


Рис.ХХХ. Реконфігурація гиперкуба за рахунок надлишковості в окремих елементах.

м

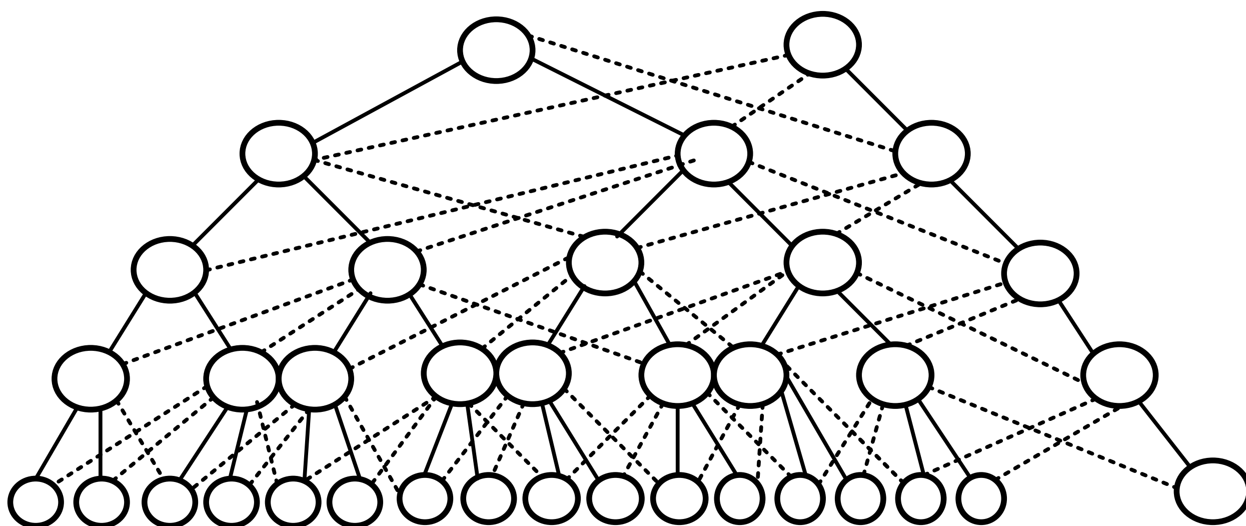


Рис.XXX. Надлишкове реконфігурируєме дерево

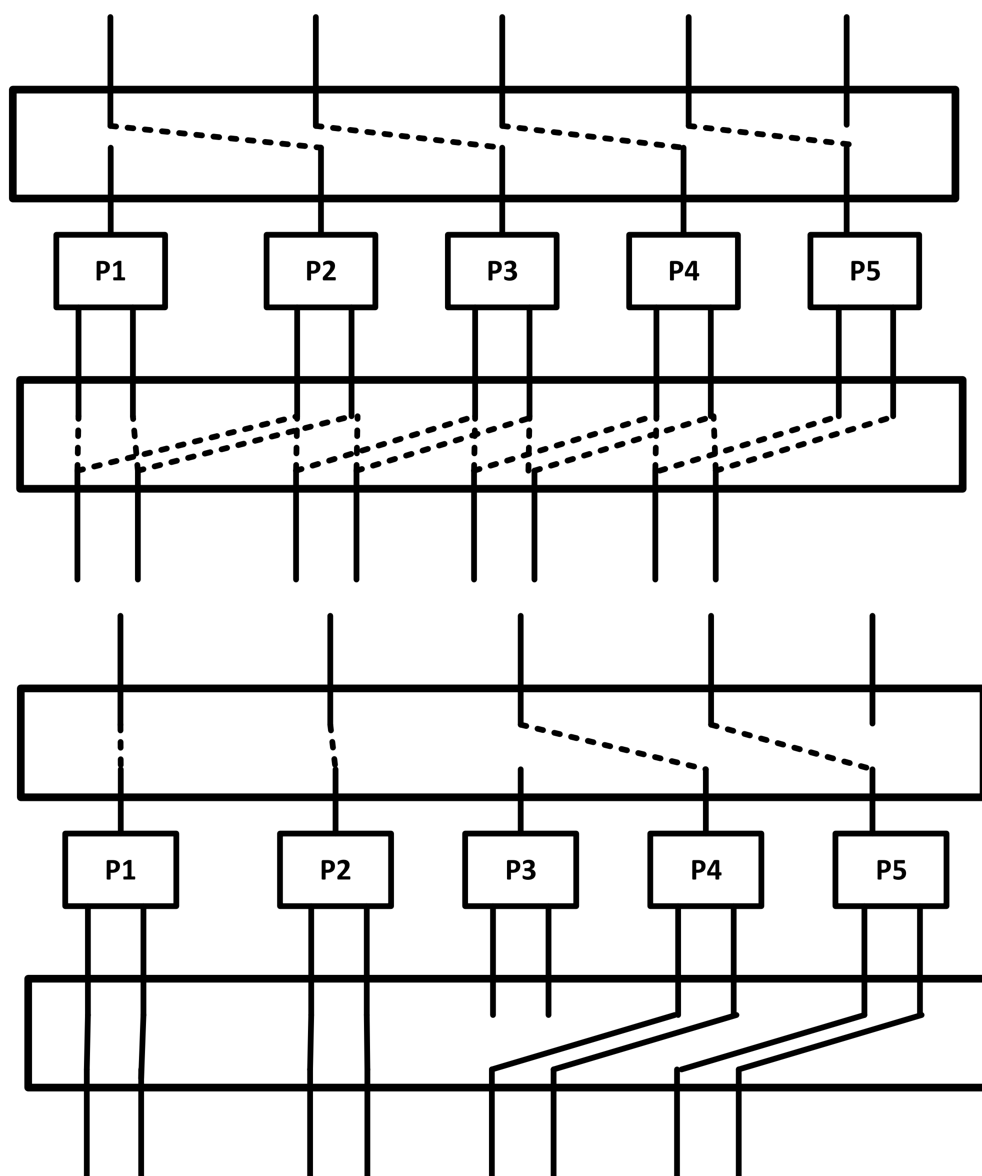


Рис.XXX. Реконфігурація у деревоподібной мережі.

4	6	8	3	20
1	6	4	7	18
2	4	5	8	19
6	3	4	1	14
13	18	21	19	71

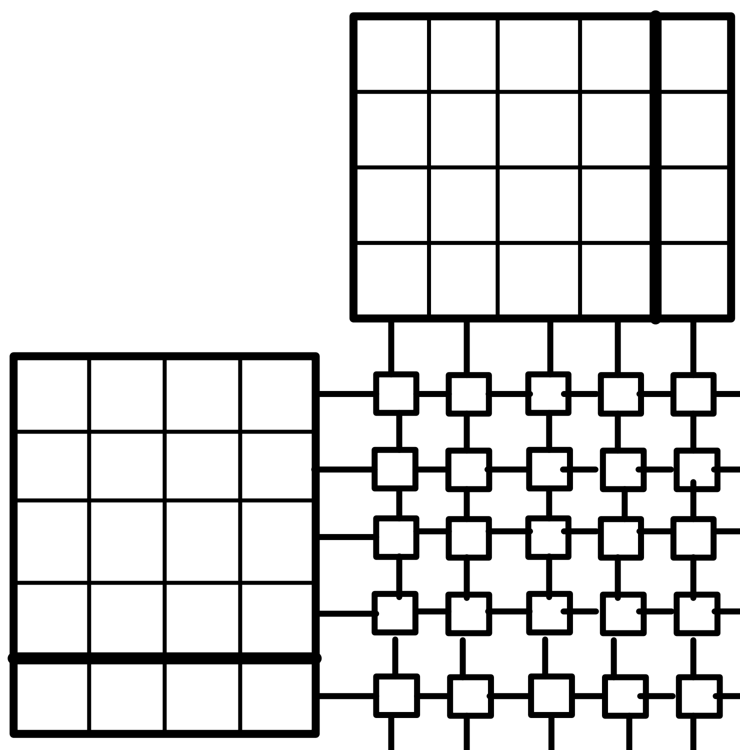
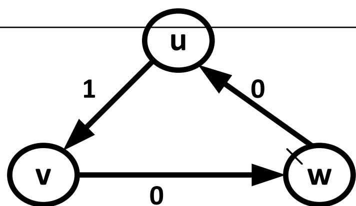
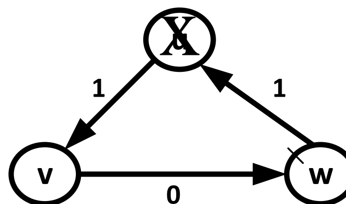
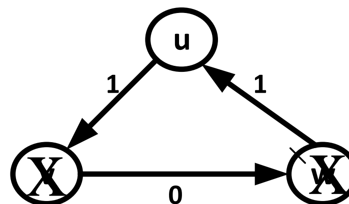


Рис.XXX. Контроль операції множення

2	1	1	0		0	1	2	3	6		5	1	2	9	17
0	2	1	1		2	-1	0	1	2		8	-1	0	3	10
3	2	1	2	x	3	0	-2	2	3	=	9	3	8	11	31
2	2	0	1		1	1	2	-1	3		5	1	6	7	19
7	7	3	4								27	4	10	30	77







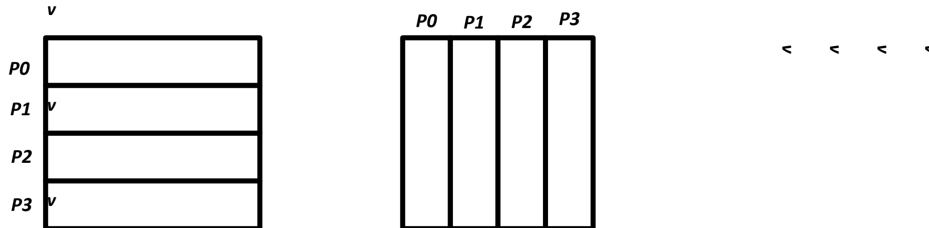
Дефектні вузли	(u,v)	(v,w)	(w,u)
немає	0	0	0
u	x	0	1
v	1 x	x	0
w	0	1	x
u,v	x	x	1
v,w	1	x	x
w,u	x	1	x
u,v,w	x	x	x

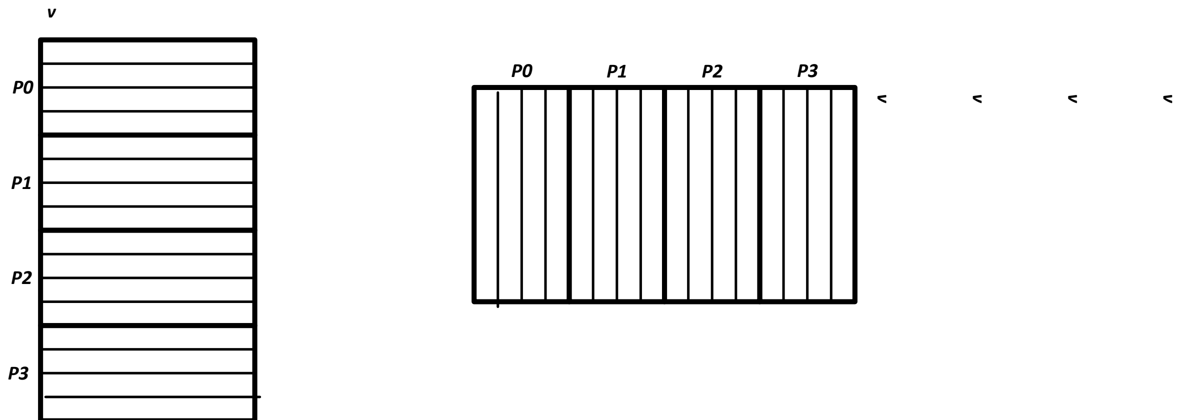
v

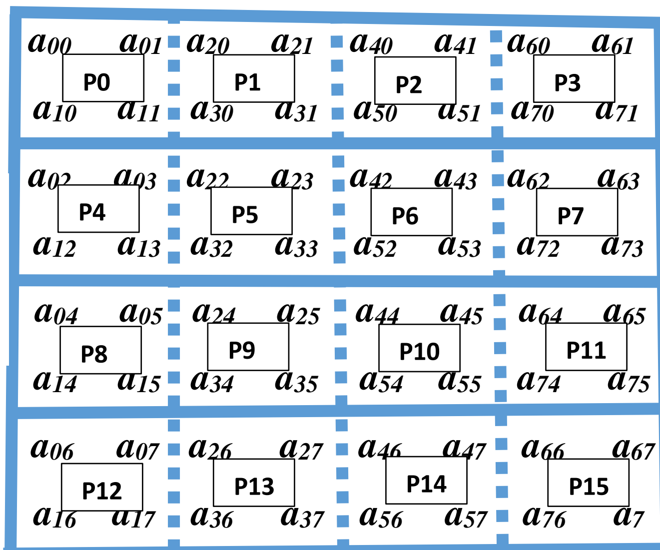
v

v

v







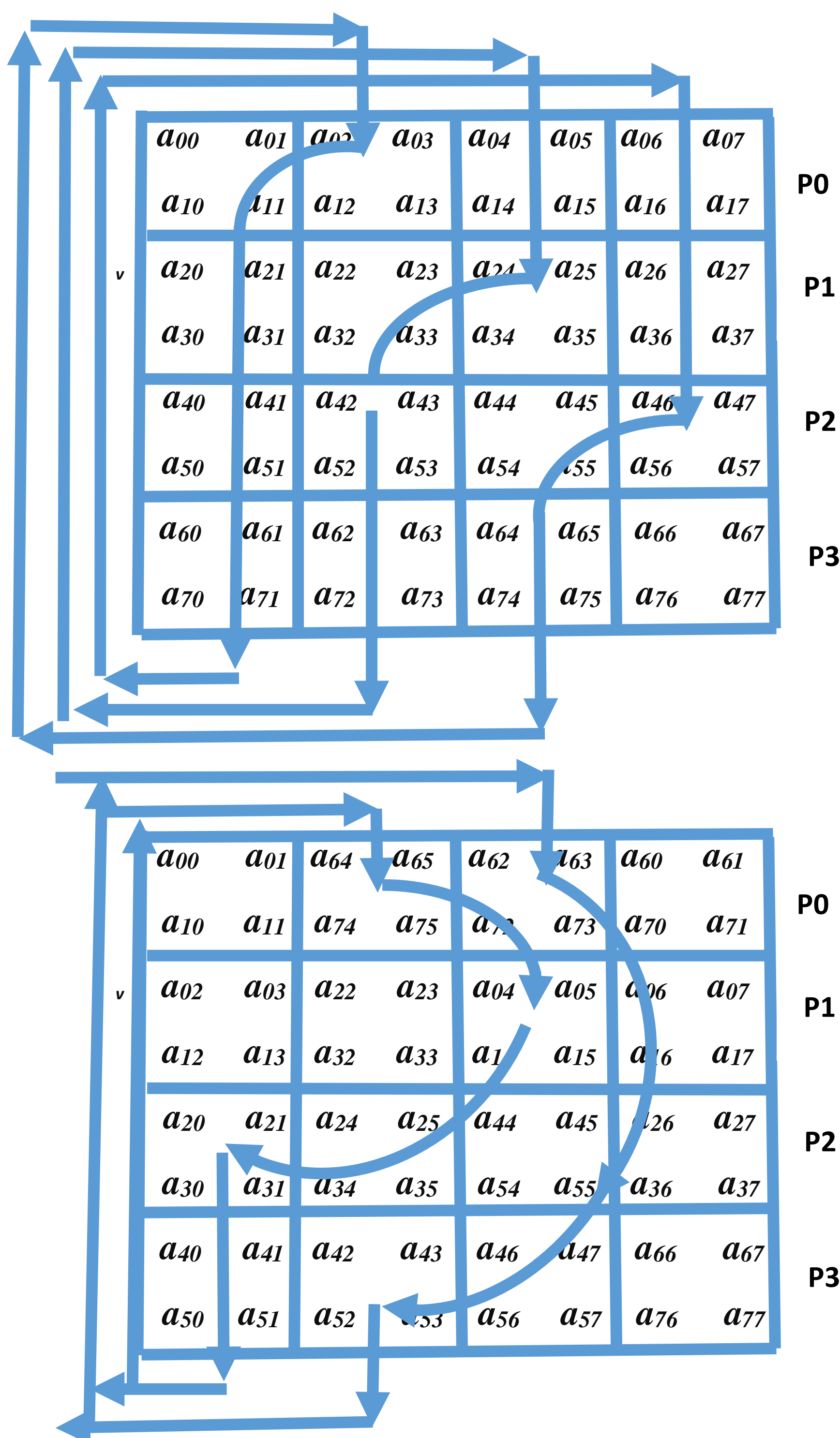


Рис.XXX. Паралельне транспонування матриці на основі алгоритму усі-всім з персональним призначенням при поділу матриці на рядки – кроки 1 і 2.

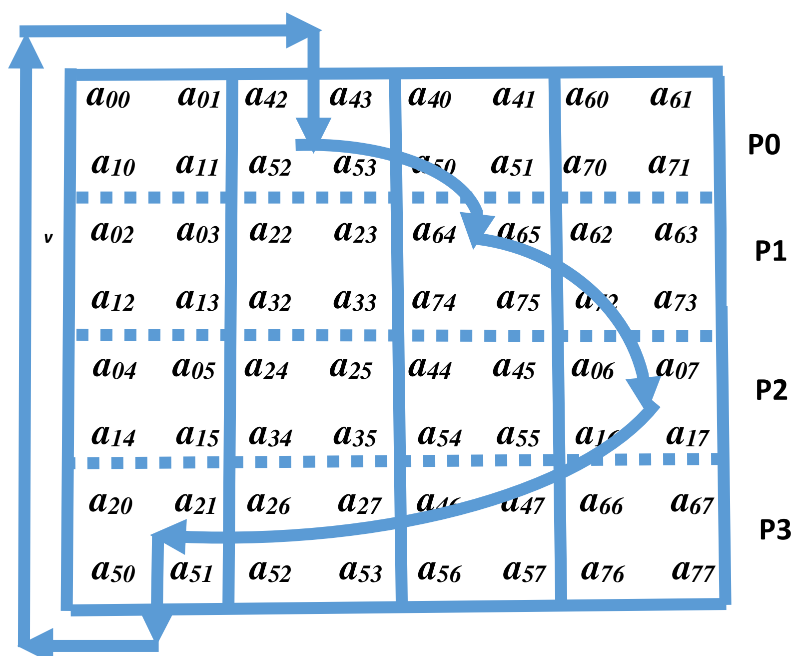


Рис.XXX. Транспонування матриці на основі алгоритму усі-всім з персональним призначенням при поділу матриці на рядки – крок 3.

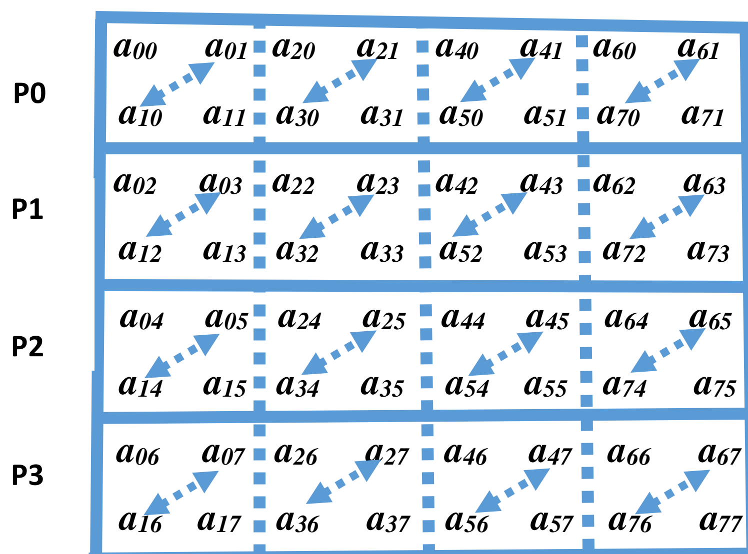
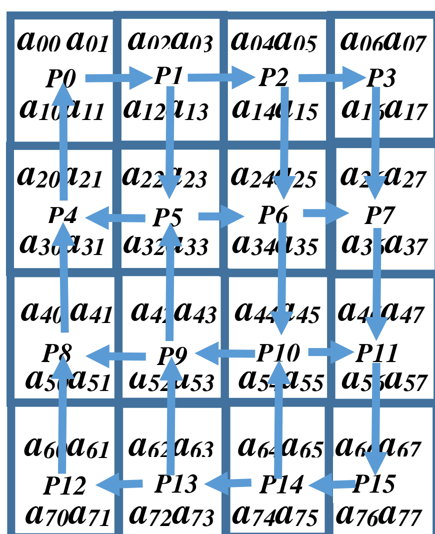
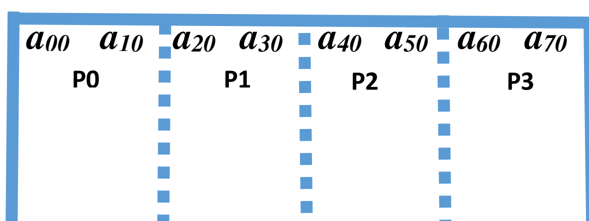


Рис.XXX. Транспонування матриці на основі алгоритму усі-всім з персональним призначенням при поділу матриці на рядки – крок 4.

P0	a_{00}	a_{10}	a_{20}	a_{30}	a_{40}	a_{50}	a_{60}	a_{70}
	a_{01}	a_{11}	a_{21}	a_{31}	a_{41}	a_{51}	a_{61}	a_{71}
P1	a_{02}	a_{12}	a_{22}	a_{32}	a_{42}	a_{52}	a_{62}	a_{72}
	a_{03}	a_{13}	a_{23}	a_{33}	a_{43}	a_{53}	a_{63}	a_{73}
P2	a_{04}	a_{14}	a_{24}	a_{34}	a_{44}	a_{54}	a_{64}	a_{74}
	a_{05}	a_{15}	a_{25}	a_{35}	a_{45}	a_{55}	a_{65}	a_{75}
P3	a_{06}	a_{16}	a_{26}	a_{36}	a_{46}	a_{56}	a_{66}	a_{76}
	a_{07}	a_{17}	a_{27}	a_{37}	a_{47}	a_{57}	a_{67}	a_{77}

Рис.XXX. Кінцевий результат транспонування матриці на основі алгоритму усі-всім з персональним призначенням при поділу матриці на рядки.



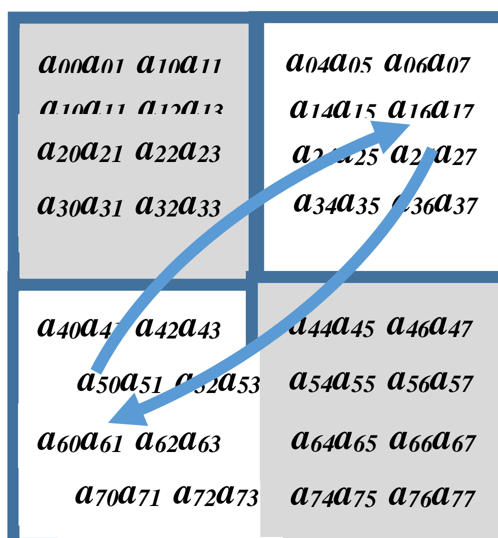


a_{01} a_{11}	a_{21} a_{31}	a_{41} a_{51}	a_{61} a_{71}
a_{02} a_{12}	a_{22} a_{32}	a_{42} a_{52}	a_{62} a_{72}
P4	P5	P6	P7
a_{03} a_{13}	a_{23} a_{33}	a_{43} a_{53}	a_{63} a_{73}
a_{04} a_{14}	a_{24} a_{34}	a_{44} a_{54}	a_{64} a_{74}
P8	P9	P10	P11
a_{05} a_{15}	a_{25} a_{35}	a_{45} a_{55}	a_{65} a_{75}
a_{06} a_{16}	a_{26} a_{36}	a_{46} a_{56}	a_{66} a_{76}
P12	P13	P14	P15
a_{07} a_{17}	a_{27} a_{37}	a_{47} a_{57}	a_{67} a_{77}

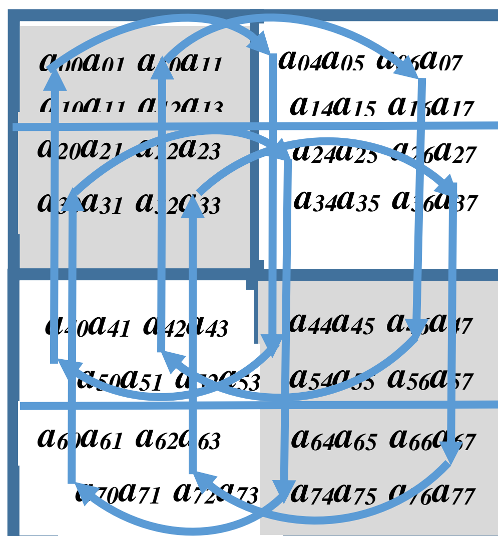
P4

P4

Рис.XXX. Паралельна реалізація операції транспозиції в меш-топології.



v



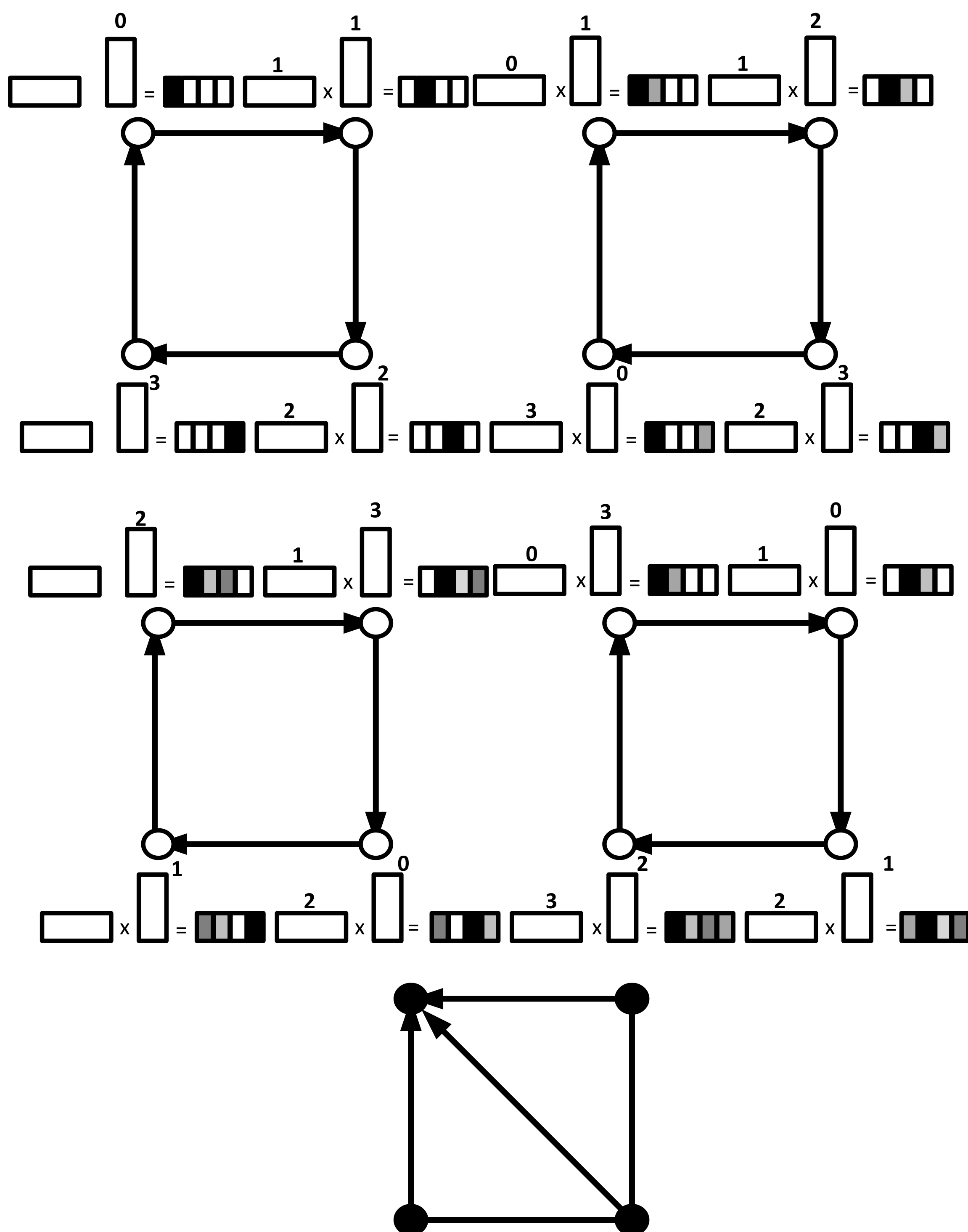


Рис.XXX. Етапи обчислення добутк матриць в гиперкубі.

Топологічні організації сучасних суперкомп'ютерів

Розглянемо топологічні організації і способи комутації, які застосовуються в сучасних суперкомп'ютерах. Для цього звернемося до рейтингу топ500.

Станом на листопад 2019 року на першому місці рейтингу знаходиться американський **суперкомп'ютер Summit** - IBM Power System AC922, який знаходиться в національній лабораторії Oak Ridge міністерства енергетики США (Теннесі, США).

Рис 3.14. Суперкомп'ютер Summit для атомних досліджень

Суперкомп'ютер складається з 4608 обчислювальних вузлів з продуктивністю в 42 Терафлопси кожен. Таким чином загальна продуктивність складає близько 192 Петафлопси. Кожен обчислювальний вузол складається з двох 22 ядерних процесорів IBM Power9, 6 відеопроцесорів Nvidia GV100 та має 512 Гб оперативної пам'яті та 96 Гб відеопам'яті. Для зв'язку вузлів в безпосередньо-зв'язану мережу використовується адаптери зв'язку точка-точка Dual Rail EDR-IB (25 GB/s) фірми Mellanox, які побудовані на технології InfiniBand. В якості топологічної оргнізації використовується Non-blocking Fat Tree - жирне дерево без блокування. Розглянемо дану топологію більш детально.

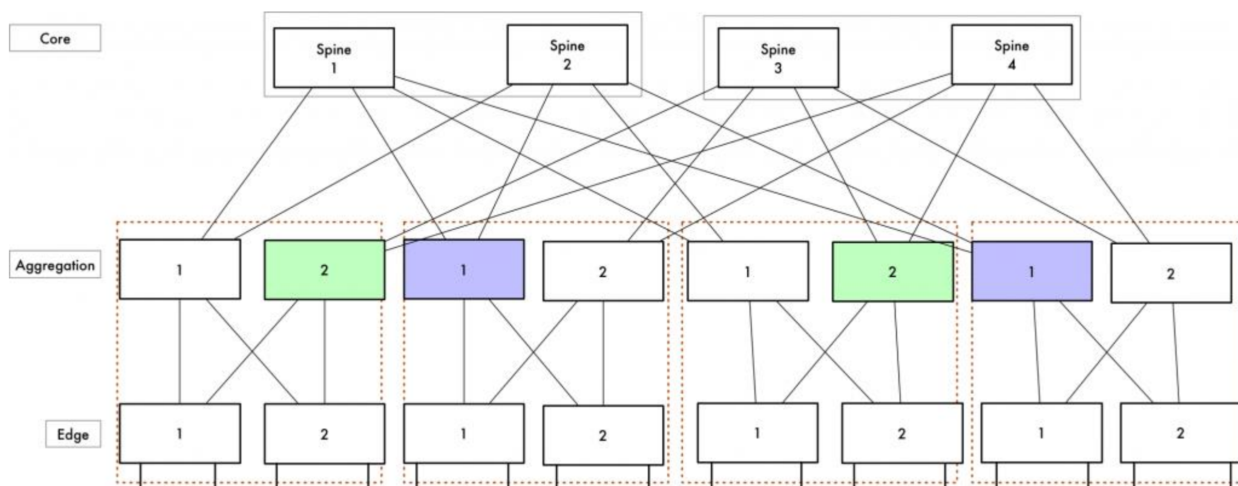


Рис 3.15. Жирне дерево без блокувань

Використання топології жирного дерева і комутації на основі віртуальних каналів дозволяє суперкомп’ютеру показати результат в 148 Петафлопсів у тесті LINPACK, що складає майже 77% від пікової продуктивності. Перевагою топології жирного дерева є багаторазове резервування всіх наявних зв’язків, що дозволяє майже гарантовано уникнути повних блокувань при передачі даних, навіть в умовах відмов компонентів комп’ютерної системи.

Розглянемо ще два варіанти топологічних організацій суперкомп’ютерів, які в свій час були одними з лідерів рейтингу топ 500, а станом на листопад 2019 входять до першої двадцятки.

Суперкомп’ютер **Trinity** - Cray XC40, який знаходиться в національній лабораторії LOS ALAMOS (США) займає 7-е місце в рейтингу топ 500 за листопад 2019 року. Trinity складається з 14 тис. вузлів, кожен з яких має 68 ядер. Загальна продуктивність суперкомп’ютера складає 20 Петафлопсів. Trinity побудований на основі топології Dragonfly (рис 3.16.)

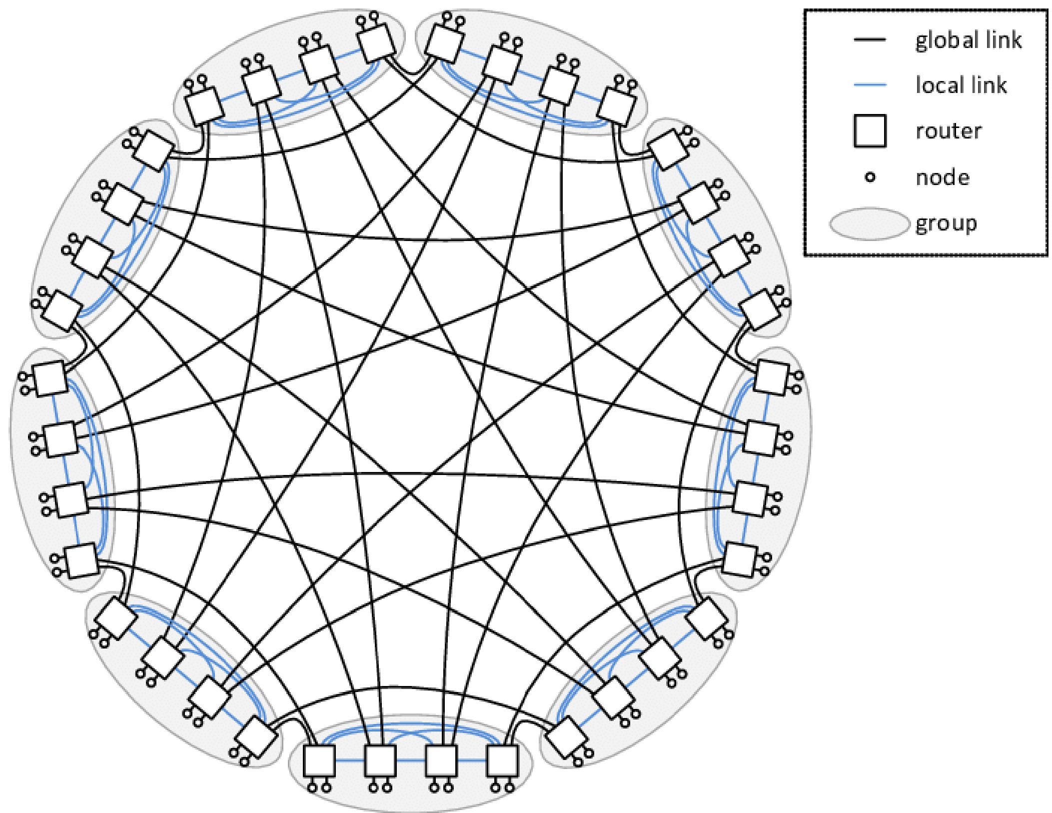


Рис 3.16 Топологія Dragonfly з параметрами h=2 (p=2, a=4), 36 маршрутизаторів і 72 процесорних вузла.

Топологія Dragonfly має наступні параметри:

p - кількість процесорів, які підключаються до одного маршрутизатора

$a-1$ - кількість лінків для кожного елемента в середині кластера. Мінус одиниця вводиться, для виключення петлі, тобто зв'язку з самим собою.

h – кількість міжкластерних лінків для кожного маршрутизатора.

Таким чином, кількість лінків у кожному роутері, тобто ступінь топології, дорівнює $s = p + a - 1 + h$. Кожен кластер містить ap процесорів, та ah міжкластерних лінків.

Загальна кількість процесорів дорівнює $N = ap(ah + 1)$.

Особливістю топологічної організації Dragonfly є використання повнозв'язних кластерів, кожен елемент якого має зв'язки з декількома дзеркальними елементами інших кластерів. Таким чином, між кластерна організація теж є повнозв'язною. В якості системи зв'язку точка-точка

використовується технологія Aries interconnect. Кожен адаптер може з’єднати 8 обчислювальних вузлів з іншими 40 мережевими адаптерами. При цьому, швидкість передачі буде складати до 5.25 GB/s на кожен порт. Кожен маршрутизатор підтримує зв’язки всередині і між кластерами.

Розглянемо суперкомп’ютер **Cray Titan** який займав перше місце в топ 500 від 2012 року. Цей суперкомп’ютер містить 18 тис. вузлів, що разом складають 552 тис. процесорів, а його загальна продуктивність складає 18 Петафлопсів.

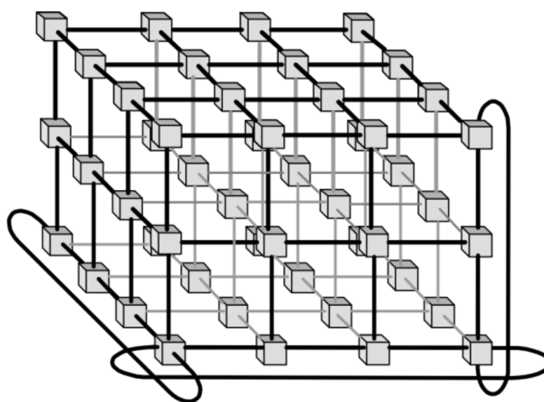


Рис. 3.17. Топологія 3D Torus для суперкомп’ютера Titan

Titan, як і суперкомп’ютер Summit, що прийшов йому на зміну, зібраний в національній лабораторії Oak Ridge міністерства енергетики США. Titan використовує топологічну організацію 3D Tor (рис 3.17.). Titan побудований на основі технології Gemini Network for Cray’s. Кожен маршрутизатор містить 48 портів, а кожен лінк містить 12 каналів, які дозволяють передавати дані зі швидкістю до 4.68GB/sec на кожне підключення.

В червні 2020 року перше місце в рейтингу суперкомп’ютерів зайняла японська машина виробництва корпорації Fujitsu. Суперкомп’ютер **Fugaku** складається з 158976 вузлів і має швидкість більше 400 Петафлопсів. Для з’єднання вузлів використовується архітектура Tofu Interconnect D, яка являє собою 6-просторовий меш (6D-тор). Це дозволяє підвищити відмовостійкість зв’язків між вузлами машини і проектувати суперкомп’ютери з швидкістю від 10 Петафлопсів [x] Ajima, Yuichiro, Shinji Sumimoto, and Toshiyuki Shimizu. "Tofu:

A 6D mesh/torus interconnect for exascale computers." Computer 11 (2009): 36-40. Розглянемо переваги даної топології більш детальніше.

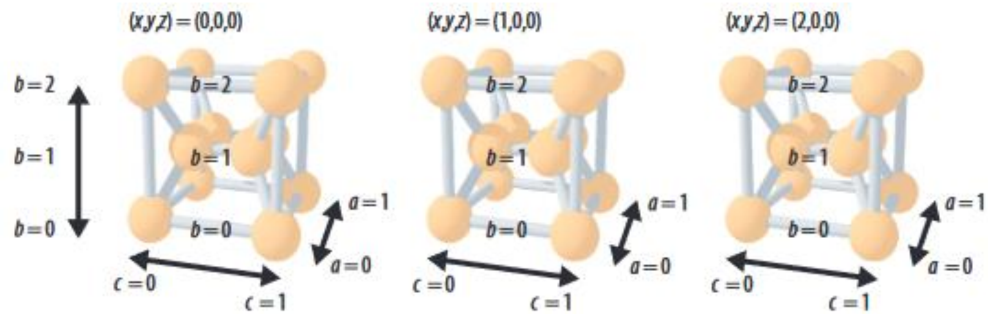


Рис 1. Дванадцять вузлів, що мають однакові координати xyz, становлять групу вузлів і з'єднані між собою осями abc [x].

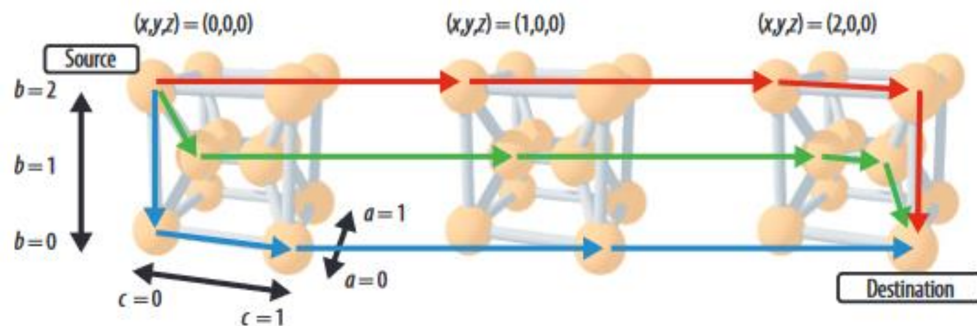


Рис 2. Маршрути від вузла (0,0,0,0,2,0) до вузла (2,0,0,0,0,1) [x].

На рис. 2 показано 3 можливі шляхи між вузлами (0,0,0,0,2,0) та (2,0,0,0,0,1). Всього таких маршрутів може бути 12 [x].

Завдання на курсовий проект.

1. Синтезувати оригінальну топологічну організацію з числом вузлів $N = 16$ і показати способи її довільного розширення.
2. Визначити характеристики синтезованої топології, до яких у першу чергу слід віднести:
ступінь S топології, діаметри D топології, топологічний трафік F.
3. Порівняти характеристики синтезованої топології з характеристиками гіперкуба відповідного порядку.
4. Виділити переваги синтезованої топології.
5. Відобразити синтезовану топологію на гіперкуб.
6. Розробити алгоритми маршрутизації для синтезованої топології:
   - один-одному (one-to-one);
   - всі-всім з персональним призначенням (all-to-all-personalized);
   - оригінальний алгоритм one-to-allдля випадків:
     - всі елементи працездатні;
     - частина елементів від 1 до $\log_2 N$ не працездатні.
7. Завантажити алгоритм деякої великої задачі в синтезовану топологію і показати послідовність її ефективного рішення.
8. Порівняти час її рішення з часом рішення в однопроцесорному варіанті.