**МІНІСТЕРСТВО ОСВІТИ І НАУКИ, МОЛОДІ ТА СПОРТУ УКРАЇНИ**
**ДВНЗ «УЖГОРОДСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ»**
**Факультет інформаційних технологій**
**Кафедра інформаційних управляючих систем та технологій**

# КОМП'ЮТЕРНІ СИСТЕМИ ТА МЕРЕЖІ: ПРОГРАМНА ПІДТРИМКА

**Навчально-методичний посібник**

**УЖГОРОД – 2016**

**Комп'ютерні системи та мережі: програмна підтримка**: методичні рекомендації до вивчення курсу для студентів спеціальностей «Інформаційні управляючі системи та технології» та «Програмне забезпечення систем» факультету інформаційних технологій УжНУ / Розробник: О.М. Левчук. – Ужгород: Видавництво УжНУ, 2016. – 64 с.

Методичний посібник з курсу «Комп'ютерні системи та мережі: програмна підтримка» містить деякі теоретичні відомості, опис предмета навчальної дисципліни у відповідності до болонського процесу, навчально-тематичний план дисципліни, зміст лекційних тем курсу, тем лабораторних занять та тем самостійної роботи студентів, перелік питань для модульних контролів, на залік, літературу.

**Розробник:** Левчук О.М., к.т.н., доцент кафедри інформаційних управляючих систем та технологій факультету інформаційних технологій УжНУ.

**Рецензенти:**

- Міца О.В., к.т.н., доцент кафедри інформаційних управляючих систем і технологій факультету інформаційних технологій УжНУ;
- Коцовський В.М., к.т.н., доцент кафедри інформаційних управляючих систем і технологій факультету інформаційних технологій УжНУ.

**Рекомендовано** кафедрою інформаційних управляючих систем та технологій
Протокол № 7 від 2 лютого 2016 року.

**Рекомендовано** Вченою радою факультету інформаційних технологій.
Протокол №__ від "___" _____ 2016 року.

# Вступ

В посібнику розглядаються основні поняття комп'ютерних систем та мереж, їх класифікації, характеристики; паралельних комп'ютерів; застосовування технології паралельного програмування при роботі з сучасними комп'ютерними системами; використання прикладних протоколів при проектуванні систем телекомунікаційних послуг.

Дано опис предмета навчальної дисципліни у відповідності до болонського процесу, навчально-тематичний план дисципліни «Комп'ютерні системи та мережі: програмна підтримка», зміст лекційних тем курсу, тем лабораторних занять та тем самостійної роботи студентів, а також перелік питань на модуль І і на залік.

## Мета і завдання курсу

Метою курсу є ознайомлення з архітектурою сучасних паралельних обчислювальних систем різного рівня складності, освоєння ними базових відомостей про поширені технології паралельного програмування у т.ч. з мовами та системами програмування, що використовуються під час їх використання, приклади застосування.

**Завдання:**

- вивчити принципи організації паралельних систем і мереж:
- ознайомитися з особливостями традиційних та перспективних технологій паралельного програмування обчислювальних систем;
- засвоїти програмне забезпечення та методи управління мережами та принципами їх адміністрування;
- вміти застосовувати технічні засоби та програмне забезпечення, що використовуються при проектуванні сучасних мереж.

# Опис предмета навчальної дисципліни

<table>
<tr><td rowspan="2">Найменування показників</td><td rowspan="2">Галузь знань, напрям підготовки, освітньо-кваліфікаційний рівень</td><td colspan="2">Характеристика навчальної дисципліни</td></tr>
<tr><td>денна форма навчання</td><td>заочна форма навчання</td></tr>
<tr><td>Кількість кредитів – 6</td><td>Галузь знань<br>0501 Інформатика та обчислювальна техніка<br>(шифр і назва)</td><td colspan="2">За вибором</td></tr>
<tr><td>Модулів – 1</td><td rowspan="4">Спеціальність (професійне спрямування):<br>8.05010301 "Програмне забезпечення систем",<br>8.05010101 "Інформаційні управляючі системи та технології"</td><td colspan="2">Рік підготовки:</td></tr>
<tr><td>Змістових модулів – 2</td><td>2016-й</td><td>2016-й</td></tr>
<tr><td></td><td colspan="2">Семестр</td></tr>
<tr><td>Загальна кількість годин - 216</td><td>9-й</td><td>9-й</td></tr>
<tr><td rowspan="8">Тижневих годин для денної форми навчання:<br>аудиторних – 4<br>самостійної роботи студента -7</td><td rowspan="8">Освітньо-кваліфікаційний рівень:<br>магістр</td><td colspan="2">Лекції</td></tr>
<tr><td>16 год.</td><td>10 год.</td></tr>
<tr><td colspan="2">Лабораторні</td></tr>
<tr><td>20 год.</td><td>-</td></tr>
<tr><td colspan="2">Самостійна робота</td></tr>
<tr><td>72 год.</td><td>98 год.</td></tr>
<tr><td colspan="2">Вид контролю: залік</td></tr>
</table>

## Програма навчальної дисципліни

**Змістовий модуль 1. Паралельні комп'ютери та системи**

**Тема 1.** Розвиток паралельних обчислювальних систем.

**Тема 2** . Класифікація сучасних паралельних комп'ютерів та систем.

**Тема 3.** Векторно-конвейєрні комп'ютери.

**Тема 4.** Паралельні комп'ютери з загальною пам'яттю.

**Тема 5.** Обчислювальні системи з розподіленою пам'яттю.

**Тема 6.** Концепція GRID та метакомп'ютінг.

**Змістовий модуль 2. Стеки протоколів та мережеве ПЗ**

**Тема 7.** Продуктивність паралельних комп'ютерів

**Тема 8.** Сучасна технологія паралельного програмування.

**Тема 9.** Стеки протоколів та типи обчислювальних мереж.

**Тема 10.** Функції мережевого програмного забезпечення.

## Структура навчальної дисципліни

| Назви змістових модулів і тем | Кількість годин | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | денна форма | | | | Заочна форма | | | |
| | усього | у тому числі | | | усього | у тому числі | | |
| | | л | лаб | с.р. | | л | лаб | с.р. |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| **Модуль 1** | | | | | | | | |
| **Змістовий модуль 1. Паралельні комп'ютери та системи** | | | | | | | | |
| Тема 1. Вступ. Актуальність проблеми створення та розвитку паралельних обчислювальних систем | 3 | 1 | | 2 | | 1 | | 6 |
| Тема 2. Класифікація сучасних паралельних комп'ютерів та систем | 8 | 2 | | 6 | | 1 | | 10 |
| Тема 3. Векторно-конвейєрні комп'ютери | 7 | 1 | | 6 | | 1 | | 10 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Тема 4. Паралельні комп'ютери з загальною пам'яттю | 7 | 1 | | 6 | | 1 | | 10 |
| Тема 5. Обчислювальні системи з розподіленою пам'яттю | 7 | 1 | | 6 | | 1 | | 10 |
| Тема 6. Концепція GRID та метакомп'ютінг | 16 | 2 | 6 | 8 | | 1 | | 12 |
| Разом за змістовим модулем 1 | 48 | 8 | 6 | 34 | | 6 | | 58 |
| **Змістовий модуль 2. Стеки протоколів та мережеве ПЗ** | | | | | | | | |
| Тема 7. Продуктивність паралельних комп'ютерів | 10 | 2 | 2 | 6 | | 1 | | 10 |
| Тема 8. Сучасна технологія паралельного програмування: використання традиційних послідовних мов, систем програмування на основі передачі повідомлень, Т-система, система НОРМА | 20 | 2 | 4 | 14 | | 1 | | 10 |
| Тема 9. Стеки протоколів та типи обчислювальних мереж | 16 | 2 | 4 | 10 | | 1 | | 10 |
| Тема 10. Функції мережевого програмного забезпечення | 14 | 2 | 4 | 8 | | 1 | | 10 |
| Разом за змістовим модулем 2 | 60 | 8 | 14 | 38 | | 4 | | 40 |
| **Усього за семестр** | **108** | **16** | **20** | **72** | **108** | **10** | | **98** |

**Теми лабораторних занять**

| № з/п | Назва теми | Кількість годин |
|---|---|---|
| 1 | Концепція GRID та метакомп'ютінг | 6 |
| 2 | Продуктивність паралельних комп'ютерів | 2 |
| 3 | Сучасна технологія паралельного програмування: використання традиційних послідовних мов, систем програмування на основі передачі повідомлень, Т-система, система НОРМА | 4 |
| 4 | Стеки протоколів та типи обчислювальних мереж | 4 |
| 5 | Функції мережевого програмного забезпечення | 4 |
| **Разом** | | **20** |

## Самостійна робота, індивідуальні завдання

| № з/п | Назва теми | Кількість годин |
|---|---|---|
| 1. | Класифікація сучасних паралельних комп'ютерів та систем | 6 |
| 2. | Векторно-конвейєрні комп'ютери | 6 |
| 3. | Паралельні комп'ютери з загальною пам'яттю | 6 |
| 4. | Обчислювальні системи з розподіленою пам'яттю | 6 |
| 5. | Концепція GRID та метакомп'ютінг | 8 |
| 6. | Продуктивність паралельних комп'ютерів | 6 |
| 7. | Сучасна технологія паралельного програмування: використання традиційних послідовних мов, систем програмування на основі передачі повідомлень, Т-система, система НОРМА | 16 |
| 8. | Стеки протоколів та типи обчислювальних мереж | 10 |
| 9. | Функції мережевого програмного забезпечення | 8 |
| | **Разом** | **72** |

## Методи навчання

Протягом навчання студенти вивчають один модуль з дисципліни. Модуль складається з двох змістових модулів. Після виконання кожного змістового модуля (лекції, практичні заняття) здійснюється поточний контроль у вигляді тестування або самостійної контрольної роботи. Студенти, які не відвідували лекції або не в повному обсязі виконали практичні завдання, до поточного контролю за змістовий модуль не допускаються.

## Методи контролю

Модульний контроль (тестування, контрольна робота, колоквіум).

## Розподіл балів, які отримують студенти

| Поточне тестування та самостійна робота | | | | | | | | | | | | **Залік** | **Індивід та самост. робота студента** | **Сума** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Змістовий модуль №1 | | | | | | Індивід та самост. робота студента | Змістовий модуль № 2 | | | | Індивід та самост. робота студента | | | |
| Т1 | Т2 | Т3 | Т4 | Т5 | Т6 | 30 | Т7 | Т8 | Т9 | Т10 | 30 | 70 | 30 | 100 |
| 10 | 10 | 10 | 10 | 15 | 15 | | 15 | 15 | 20 | 20 | | | | |

Т1, Т2 ... Т10 – теми змістових модулів.

## Шкала оцінювання: національна та ECTS

| Сума балів за всі види навчальної діяльності | Оцінка ECTS | Оцінка за національною шкалою | |
|---|---|---|---|
| | | для екзамену, курсового проекту (роботи), практики | для заліку |
| 90 – 100 | **A** | відмінно | зараховано |
| 82-89 | **B** | добре | |
| 74-81 | **C** | | |
| 64-73 | **D** | задовільно | |
| 60-63 | **E** | | |
| 35-59 | **FX** | незадовільно з можливістю повторного складання | не зараховано з можливістю повторного складання |
| 0-34 | **F** | незадовільно з обов'язковим повторним вивченням дисципліни | не зараховано з обов'язковим повторним вивченням дисципліни |

## Методичне забезпечення

1. М.В. Макарова Інформатика і комп'ютерна техніка: Навчальний посібник / М.В. Макарова, Г. В. Карнаухова, С.В. Запара. 2-ге видання. - Суми: Університетська книга, 2008.
2. Д. О.Рзаєв та інші. " Інформатика та комп'ютерна техніка "Навчально – методичний посібник для самостійного вивчення дисципліни – К: КНЕУ, 2003.

## Рекомендована література


1. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. – СПб.: БХВ-Петербург, 2002.– 608 с.: іл.
2. Норенков И.П. Основы автоматизированного проектирования: Учеб. для вузов 2-е изд., М.:Изд-во МГТУ им. Н.Э. Баумана, 2002.–336 с.: іл.
3. Мельник А.О. Архітектура комп'ютера. Наукове видання. Луцьк: Волинська обласна друкарня, 2008.– 470 с.
4. Компьютерные сети. Принципы, технологии, протоколы / В.Г. Олифер, Н.А. Олифер. – СПб.: Питер, 2001. – 672 с.
5. Э. Таненбаум – Компьютерные сети.: Питер. 2003. – 992 с.
6. Алиев Т.И. – Сети ЭВМ и телекоммуникации.: СПбГУ ИТМО. 2011. – 400с.
7. М.Й. Павликевич. Мережі типу Ethernet. Львів, 2003. – 75 с.
8. В. Ф. Мелехин, Е. Г. Павловський – Вычислительные машины, системы и сети. – Москва: Издательство «Академия», 2007. – 560 с.
9. Довідник по інформаційним технологіям: http://www.mark-itt.ru.
10. Інтернет-університет інформаційних технологій: http://www.intuit.ru.

**Перелік питань на модуль**

1. Передумова створення паралельних обчислювальних систем
2. Паралельні обчислювальні системи
3. Моделі паралельних комп’ютерів
4. SIMD архітектура
5. MIMD архітектура
6. Комп'ютери з розподіленою пам'яттю
7. Комп'ютери з загальною пам'яттю
8. Комп’ютери з віртуально спільною пам’яттю
9. Архітектура багатопроцесорних обчислювальних систем: основні напрямки розвитку суперкомп'ютерних технологій
10. Векторно-конвеєрні суперкомп'ютери
11. Симетричні мультипроцесорні системи (SMP)
12. Системи з масовим паралелізмом (MPP)
13. Кластерні системи
14. Стеки протоколів
15. Стек протоколів OSI
16. Стек протоколів TCP/IP
17. Стек протоколів IPX/SPX
18. Мережеве обладнання
19. Системне мережеве програмне забезпечення
20. Прикладне мережеве програмне забезпечення

**Перелік питань на залік**

1. Передумова створення паралельних обчислювальних систем
2. Паралельні обчислювальні системи
3. Моделі паралельних комп’ютерів
4. SIMD архітектура
5. MIMD архітектура
6. Комп'ютери з розподіленою пам'яттю
7. Комп'ютери з загальною пам'яттю
8. Комп’ютери з віртуально спільною пам’яттю
9. Архітектура багатопроцесорних обчислювальних систем: основні напрямки розвитку суперкомп'ютерних технологій
10. Векторно-конвеєрні суперкомп'ютери
11. Симетричні мультипроцесорні системи (SMP)
12. Системи з масовим паралелізмом (MPP)
13. Кластерні системи
14. Стеки протоколів
15. Стек протоколів OSI
16. Стек протоколів TCP/IP
17. Стек протоколів IPX/SPX
18. Мережеве обладнання
19. Системне мережеве програмне забезпечення
20. Прикладне мережеве програмне забезпечення

# 1. Основні поняття паралельних обчислювальних систем.

У міру того, як комп'ютери стають усе більш швидкими, може виникнути думка, що комп'ютери, у кінцевому рахунку, стануть "досить швидкими", і що потреба збільшення обчислювальної потужності буде поступово зменшуватися. Однак історія розвитку комп'ютерів показує, що в міру того як нова технологія задовольняє уже відомі прикладні задачі, з'являються нові, інтерес до яких був викликаний цією технологією і які тепер вимагають розробки ще більш нової технології і так далі. Так, наприклад, перші дослідження ринку збуту фірмою Cray Research пророкували ринок у десяток суперкомп'ютерів, однак з тих пір було продано тисячі суперкомп'ютерів.

Традиційно, збільшення обчислювальної потужності мотивувалося числовими моделюваннями складних систем, таким як автомобілебудування, нафто- і газовидобування, фармакологія, прогноз погоди і моделювання зміни клімату, сейсмологічна розвідка, проектування електронних та механічних пристроїв, синтез нових матеріалів, виробничі процеси, фізичні і хімічні процеси. Однак, на сьогоднішній день найбільш істотними силами, що вимагають розробки більш швидких комп'ютерів, стають комерційні додатки, для яких необхідно, щоб комп'ютер був здатний обробити величезні об'єми даних, причому використовуючи різноманіття складних методів. Ці додатки, включають бази даних (особливо, якщо вони використовуються при прийнятті рішень), відеоконференції, спільні робітничі середовища, автоматизацію діагностування в медицині, розвинуту графіку і віртуальну реальність (особливо для промисловості розваг).

Хоча комерційні додатки можуть у достатній мірі визначити архітектуру більшості майбутніх паралельних комп'ютерів, традиційні наукові додатки будуть залишатися важливими споживачами паралельних обчислювальних технологій. Дійсно, оскільки нелінійні ефекти ускладнюють розуміння теоретичних досліджень, експерименти стають усе більш і більш дорогими, непрактичними чи неможливими з політичних чи яких-небудь інших причин (наприклад, США проводить ядерні іспити, використовуючи лише суперкомп'ютери), то обчислювальні дослідження складних систем стають усе більш і більш важливими. Обчислювальні витрати, звичайно, збільшуються як четвертий ступінь і навіть більше від точності обчислень. Наукові дослідження часто характеризуються великими вимогами до обсягу пам'яті, підвищеними вимогами до організації введення-введення.

Відомо багато фактів, де використання суперкомп'ютерів допомогло уникнути великих витрат коштів, часу, людських ресурсів. Ось деякі з них:

- у 1995 році корпус автомобіля Nissan Maxima вдалося зробити на 10% міцнішим завдяки використанню суперкомп'ютера фірми Cray

(The Atlanta Journal, 28 травня, 1995г). За допомогою нього були знайдені не тільки слабкі місця кузова, але і найбільш ефективний спосіб їхнього вилучення;

- розвиток однієї з найбільших світових систем резервування Amadeus, використовуваної тисячами агентств із 180.000 терміналів у більш ніж ста країнах. Встановлення двох серверів Hewlett-Packard T600 по 12 процесорів у кожному дозволила довести ступінь оперативної приступності центральної системи до 99.85% при поточній завантаженні близько 60 мільйонів запитів у добу.

І таких прикладів можна знайти всюди. У свій час дослідники фірми DuPont шукали заміну хлорофлюорокарбону. Потрібно було знайти матеріал, що має ті ж позитивні якості: незаймистість, стійкість до корозії і низьку токсичність, але без шкідливого впливу на озоновий шар Землі. За один тиждень були проведені необхідні розрахунки на суперкомп'ютері з загальними витратами близько 5 тисяч доларів. По оцінках фахівців DuPont, при використання традиційних експериментальних методів досліджень необхідно було б біля трьох місяців і 50 тисяч доларів і це без обліку часу, необхідного на синтез і очищення необхідної кількості речовини.

Наведених факти свідчать про важливість та необхідність розвитку суперкомп'ютерів. За цим стоїть ціла низка проблем, які потрібно вирішувати і які будуть представлені нижче.

**Паралельні обчислювальні системи.**

Визначень суперкомп'ютерам намагалися давати багато, іноді серйозних, іноді іронічних. Паралельний комп'ютер - це набір процесорів, здатних спільно працювати при вирішенні обчислювальних задач. Таке визначення достатньо широке, що включає як паралельні суперкомп'ютери, що мають сотні чи тисячі процесорів, так і мережі робочих станцій. Коли ця тема піднімалася в конференції comp.parallel, Кен Батчер (Ken Batcher) запропонував такий варіант: суперкомп'ютер – це пристрій, що зводить проблему обчислень до проблеми введення/виведення. Тобто те, що раніше довго обчислювалося, іноді скидаючи щось на диск, на суперкомп'ютері може виконатися миттєво, переводячи вказівники неефективності на відносно повільні пристрої введення/виведення.

Ефективність найшвидших комп'ютерів зросла майже по експоненті. Перші комп'ютери виконували кілька десятків операцій з плаваючою комою за секунду, а продуктивність паралельних комп'ютерів середини дев'яностих досягає десятків і навіть сотень мільярдів операцій у секунду, і, швидше за все цей ріст буде продовжуватися. Однак архітектура обчислювальних систем, що визначають цей ріст, змінилася радикально - від послідовної до паралельної. Ера однопроцесорних комп'ютерів продовжувалася до появи

сімейства CRAY X-MP / Y-MP - слабко паралельних векторних комп'ютерів з 4 - 16 процесорами, яких у свою чергу перемінили комп'ютери з масовим паралелізмом, тобто комп'ютери з чи сотнями тисячами процесорів.

Ефективність комп'ютера залежить безпосередньо від часу, необхідного для виконання базової операції і кількості базових операцій, що можуть бути виконані одночасно. Час виконання базової операції обмежений часом виконання внутрішньої елементарної операції процесора (тактом процесора). Зменшення такту обмежене фізичними межами, такими як швидкість світла. Щоб обійти ці обмеження, виробники процесорів намагаються реалізувати паралельну роботу всередині чіпа - при виконанні елементарних і базових операцій. Однак теоретично було показано, що стратегія Надвисокого Рівня Інтеграції (Very Large Scale Integration - VLSI) є дорогою, що час виконання обчислень сильно залежить від розміру мікросхеми. Поряд з VLSI для підвищення продуктивності комп'ютера використовуються й інші способи: конвеєрна обробка (різні стадії окремих команд виконується одночасно), багатофункціональні модулі (окремі множники, суматори, і т.д., управляються одним потоком команд).

Все більше і більше в ЕОМ включається більше "обчислювальних блоків" і відповідна логіка їхнього з'єднання. Кожен такий "обчислювальний блок" має свої власні процесор, пам'ять. Успіхи VLSI технології в зменшенні числа компонент комп'ютера, полегшують створення таких ЕОМ. Крім того, оскільки, хоча і дуже приблизно, вартість ЕОМ пропорційна числу наявних у ній компонент, то збільшення інтеграції компонент дозволяє збільшити число процесорів в ЕОМ при не дуже значному підвищенні вартості.

Інша важлива тенденція розвитку обчислень – це величезне збільшення продуктивності мереж ЕОМ. Ще недавно мережі мали швидкодію в 1.5 Мбіт/с, сьогодні вже існують мережі зі швидкодією в декілька гігабіт за секунду. Поряд із збільшенням швидкодії мереж збільшується надійність передачі даних. Це дозволяє розробляти додатки, що використовують фізично розподілені ресурси, начебто вони є частинами одного багатопроцесорного комп'ютера. Наприклад, колективне використання вилучених баз даних, обробка графічних даних на одному чи декількох графічних комп'ютерах, а вивід і управління в реальному масштабі часу на робочих станціях.

Розглянуті тенденції розвитку архітектури і використання комп'ютерів, мереж дозволяють припустити, що у майбутньому паралельність не буде участю лише суперкомп'ютерів, вона проникне і на ринок робочих станцій, персональних комп'ютерів і мереж ЕОМ. Програми будуть використовувати не тільки безліч процесорів комп'ютера, але і процесори, доступні по мережі. Оскільки більшість існуючих алгоритмів припускають використання одного процесора, то будуть потрібні нові алгоритми і програми здатні виконувати

багато операцій одночасно. Наявність і використання паралелізму буде ставати основною вимогою при розробці алгоритмів і програм.

Число процесорів у комп'ютерах буде збільшуватися, отже, можна припустити, що протягом терміну служби програмного забезпечення воно буде експлуатуватися на кількості процесорів, яка постійно збільшується чи зменшується, залежно від потреб задачі. У цьому випадку можна припустити, що для захисту капіталовкладень у програмне забезпечення, маштабованість (scalability) програмного забезпечення (гнучкість стосовно зміни кількості використовуваних процесорів) буде настільки ж важлива як переносимість. Програма, здатна використовувати тільки фіксоване число процесорів, буде така ж недосконала, як і програма, здатна працювати тільки на одному типі комп'ютерів.

Помітною ознакою багатьох паралельних архітектур є те, що доступ до локальної пам'яті процесора дешевший, ніж доступ до віддаленої пам'яті (пам'яті інших процесорів мережі). Отже, бажано, щоб доступ до локальних даних був більш частим, ніж доступ до віддалених даних. Таку властивість програмного забезпечення називають локальністю (locality). Поряд з паралелізмом і маштабованістю, він є основною вимогою до паралельного програмного забезпечення. Важливість цієї властивості визначається відношенням вартості віддаленого і локального звертань до пам'яті. Це відношення може варіюватися від 10:1 до 1000:1 чи більше, що залежить від відносної ефективності процесора, пам'яті, мережі і механізмів, використовуваних для розміщення даних у мережі і їх добування.

## 2. Моделі паралельних комп'ютерів.

Із самого початку комп'ютерної ери існувала необхідність у все більш і більш продуктивних системах. В основному це досягалося в результаті еволюції технологій виробництва комп'ютерів. Поряд з цим мали місце спроби використовувати кілька процесорів в одній обчислювальній системі з розрахунку на те, що буде досягнуте відповідне збільшення продуктивності. Першою такою спробою, здійсненою на початку 70-х років, є ILLIAC IV. Зараз є маса паралельних комп'ютерів і проектів їх реалізації.

Архітектури паралельних комп'ютерів можуть значно відрізнятися один від одного. Розглянемо деякі істотні поняття і компоненти паралельних комп'ютерів. Паралельні комп'ютери складаються з трьох основних компонентів:

1. процесори;
2. модулі пам'яті;
3. комутаційна мережа.

Можна розглядати і більш детальнішу розбиття паралельних комп’ютерів на компоненти, однак, дані три компоненти найкраще відрізняють один паралельний комп’ютер від іншого.

Комутаційна мережа з’єднує процесори один з одним і іноді також з модулями пам’яті. Процесори, що використовуються в паралельних комп’ютерах, звичайно такі ж, як і процесори однопроцесорних систем, хоча сучасна технологія дозволяє розмістити на мікросхемі не тільки один процесор. На мікросхемі разом із процесором можуть бути розташовані ті їх складові, які дають найбільший ефект при паралельних обчисленнях. Наприклад, мікросхема трансп'ютера поряд з 32-розрядним мікропроцесором і 64-розрядним співпроцесором арифметики з плаваючою комою, містить всередині кристальний ОЗП ємністю 4 Кбайт, 32-розрядну шину пам’яті, що дозволяє адресувати до 4 Гбайт зовнішньої, стосовно кристала пам’яті, чотири послідовних двонаправлених ліній зв’язку, що забезпечують взаємодію трансп'ютера з зовнішнім світом і працюючих паралельно з ЦПП, інтерфейс зовнішніх подій.

Однією із властивостей, що розрізняють паралельні комп’ютери, є кількість можливих потоків команд. Розрізняють наступні архітектури:

1. SIMD (Single Instruction Multiple);
2. MIMD (Multiple Instruction Multiple).

**Модель SIMD.**

SIMD (Single Instruction Multiple). SIMD комп’ютер має N ідентичних синхронно працюючих процесорів, N потоків даних і один потік команд. Кожен процесор володіє власною локальною пам’яттю. Мережа, що з’єднує процесори, звичайно має регулярну топологію.

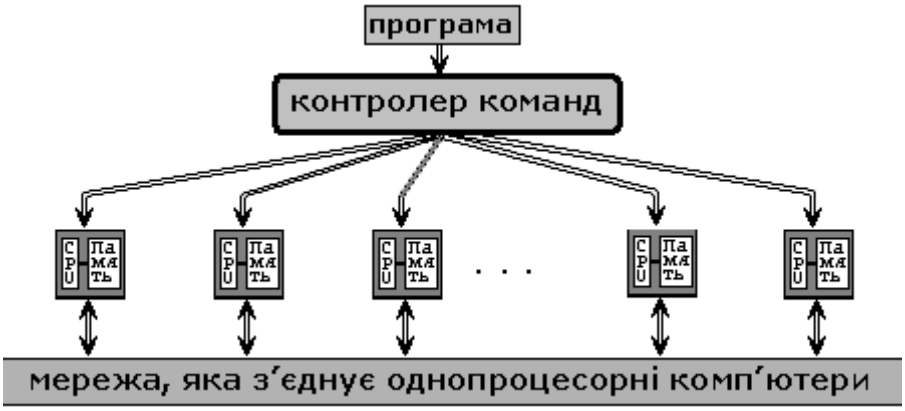



Рис.1 Модель SIMD.

Процесори інтерпретують адреси даних або як локальні адреси власної пам’яті, або як глобальні адреси, можливо, модифіковані додаванням

локальної базової адреси. Процесори одержують команди від одного центрального контролера команд і працюють синхронно, тобто на кожному кроці всі процесори виконують ту саму команду над даними з власної локальної пам’яті.

Така архітектура з розподіленою пам'яттю часто згадується як архітектура з паралелізмом даних (data-parallel), тому що паралельність досягається при наявності одиночного потоку команд, що діє одночасно на декілька частин даних.

SIMD підхід може зменшити складність як апаратного, так і програмного забезпечення, але він підходить тільки для спеціалізованих проблем, що характеризуються високим ступенем регулярності, наприклад, обробка зображення і деякі числові моделювання.

**Модель MIMD.**

MIMD (Multiple Instruction Multiple). MIMD комп'ютер має N процесорів, N потоків команд і N потоків даних. Кожен процесор функціонує під управлінням власного потоку команд, тобто MIMD комп'ютер може паралельно виконувати зовсім різні програми.

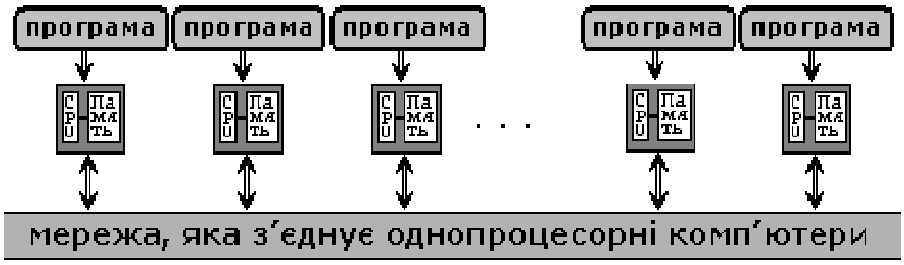



Рис. 2. Модель MIMD.

MIMD архітектури далі класифікуються в залежності від фізичної організації пам'яті, способу доступу до модулів пам’яті, тобто чи має процесор свою власну локальну пам'ять і звертається до інших блоків пам'яті, використовуючи комутаційну мережа, чи комутаційна мережа об’єднує всі процесори з загальнодоступною пам'яттю. Виходячи з доступу до пам’яті, її організації, розрізняють наступні типи паралельних (MIMD) архітектур:

1. **Комп'ютери з розподіленою пам'яттю** (distributed memory) – Кожен процесор має доступ лише до локальної, власної пам’яті. Процесори об’єднані в мережу. Доступ до віддаленої пам’яті можливий тільки за допомогою системи обміну повідомленнями. Процесор може звертатися до локальної пам'яті, може посилати й одержувати повідомлення, передані через мережу, що з'єднує процесори. Повідомлення використовуються для здійснення зв'язку між процесорами або, що є еквівалентним, для читання і

запису віддалених блоків пам'яті. В ідеалізованій мережі вартість посилки повідомлення між двома вузлами мережі не залежить як від розташування обох вузлів, так і від трафіку мережі, але залежить від довжини повідомлення.

Сюди належать так звані масово-паралельні обчислювальні (massively parallel processing – MPP) системи. Особливості даної архітектури наступні;

**Архітектура.** Система складається з однорідних обчислювальних вузлів, що включають:

- один чи декілька центральних процесорів (звичайно RISC);
- локальну пам'ять (прямий доступ до пам'яті інших вузлів неможливий);
- комунікаційний процесор чи мережевий адаптер;
- іноді - жорсткі диски чи інші пристрої введення/виведення.

До системи можуть бути додані спеціальні вузли введення/виведення і управляючі вузли. Вузли зв'язані через деяке комунікаційне середовище (високошвидкісна мережа, комутатор та інше).

**Приклади.**

- IBM RS/6000 SP2;
- Intel PARAGON/ASCIRed;
- SGI / CRAY T3E;
- Hitachi SR8000;
- системи Parsytec.

**Маштабованість.** Загальне число процесорів у реальних системах досягає декількох тисяч (ASCI Red, Blue Mountain).

**Операційна система.** Існують два основних варіанти:

- повноцінна операційна система працює тільки на управляючій машині (front-end), на кожному вузлі працює скорочений варіант операційної системи, що лише забезпечує роботу розташованої в ньому гілки паралельного додатка. (Cray T3E);
- на кожному вузлі працює повноцінна UNIX-подібна операційна система (варіант, близький до кластерного підходу). Прикладом є IBM RS/6000 SP з встановленою окремо на кожнім вузлі операційною системою AIX.

**Модель програмування.** Програмування в рамках моделіпередачі повідомлень (MPI, PVM, BSPlib)

2. **Комп'ютери з загальною пам'яттю** (True shared memory). Всі процесори спільно звертаються до загальної пам'яті, як правило, через шину чи ієрархію шин. В ідеалізованої PRAM (Parallel Random Access Machine - паралельна машина з довільним доступом) моделі, яка часто використовується в теоретичних дослідженнях паралельних алгоритмів, будь-який процесор може звертатися до будь-якої комірки пам'яті за той самий час. У таких комп'ютерах не можна істотно збільшити число процесорів, оскільки при цьому відбувається різке збільшення числа конфліктів доступу до шини. На практиці маштабованість цієї архітектури звичайно приводить до деякої форми ієрархії пам'яті. Частота звертань до загальної пам'яті може бути зменшена за рахунок збереження копій часто використовуваних даних у кеш-пам'яті, зв'язаній з кожним процесором. Доступ до цієї кеш-пам'яті набагато швидший, ніж безпосередній доступ до загальної пам'яті.

До цього класу відносяться так звані симетричні мультипроцесорні (symmetric multiprocessor SMP) системи. Особливості даної архітектури наступні;

**Архітектура.** Система складається з декількох однорідних процесорів і масиву загальної пам'яті (звичайно з декількох незалежних блоків). Всі процесори мають доступ до будь-якої частини пам'яті з однаковою швидкістю. Процесори підключені до пам'яті або за допомогою загальної шини (базові 2-4 процесорні SMP-сервери), або за допомогою комутатора (HP 9000). Апаратно підтримується когерентність кешів.

**Приклади.**

- HP 9000 V-class, N-class;
- SMP-сервери і робочі станції на базі процесорів Intel (IBM, HP, Compaq, Dell, ALR, Unisys, DG, Fujitsu та інші).

**Маштабованість.** Наявність загальної пам'яті значно спрощує взаємодію процесорів між собою, однак накладає сильні обмеження на їх число - не більше 32 у реальних системах. Для побудови маштабованих систем на базі SMP використовуються кластерні архітектури.

**Операційна система.** Вся система працює під управлінням єдиної операційної системи (звичайно UNIX-подібна, але для Intel-платформ підтримується Windows NT). Операційна система автоматично (у процесі роботи) розподіляє процеси між процесорами (scheduling), але іноді можлива і явна прив'язка.

**Модель програмування.** Програмування в моделі загальної пам'яті (POSIX threads, OpenMP). Для SMP-систем існують порівняно ефективні засоби автоматичного розпаралелювання.

3. **Комп’ютери з віртуально спільною пам’яттю** (Virtual shared memory). У таких системах загальна пам’ять, як така, відсутня. Кожен процесор має власну локальну пам’ять. Він може звертатися до локальної пам’яті інших процесорів, використовуючи “глобальну адресу”. Якщо “глобальна адреса” вказує не на локальну пам’ять, то доступ до пам’яті реалізується за допомогою повідомлень з малою затримкою, що пересилаються по мережі, яка з’єднує процесори.

До цього класу належать так звані кластерні (claster) системи. Особливості даної архітектури наступні;

**Архітектура.** Набір робочих станцій загального призначення, використовується як дешевий варіант масово-паралельного комп'ютера. Для зв'язку вузлів використовується одна із стандартних мережевих технологій (Fast/Gigabit Ethernet, Myrinet) на базі шинної архітектури чи комутатора. При об'єднанні в кластер комп'ютерів різної потужності чи різної архітектури, говорять про гетерогенні (неоднорідні) кластери. Вузли кластера можуть одночасно використовуватися в якості робочих станцій користувачів. У випадку, коли це не потрібно, вузли можуть бути істотно полегшені і встановлені у стійку.

**Приклади.**

- NT-кластер у NCSA;
- Beowulf-кластери.

**Операційна система.** Використовуються стандартні операційні системи для робочих станцій, частіше, вільно розповсюджувані - Linux/FreeBSD, разом зі спеціальними засобами підтримки паралельного програмування і розподілу навантаження.

**Модель програмування.** Програмування, як правило, у рамках моделі передачі повідомлень (частіше - MPI). Дешевизна подібних систем обертається великими накладними витратами на взаємодію паралельних процесів між собою, що сильно звужує потенційний клас розв'язуваних задач.

Відзначимо також два класи комп’ютерних систем, що іноді використовуються як паралельні комп’ютери:

- локальні обчислювальні мережі (LAN), у яких комп’ютери знаходяться у фізичній близькості і з’єднані швидкою мережею;
- глобальні обчислювальні мережі (WAN), що з’єднують географічно розподілені комп’ютери.

Хоча системи цього виду вводять додаткові властивості, такі як надійність і захист, у багатьох випадках вони можуть розглядатися як MIMD комп’ютери, хоча і з високою вартістю віддаленого доступу.

## 3. Архітектура багатопроцесорних обчислювальних систем

Експериментальні розробки зі створення багатопроцесорних обчислювальних систем почалися в 70-х роках 20 століття. Однією з перших таких систем стала розроблена в університеті Ілінойса МОС ILLIAC IV, яка включала 64 (у проекті до 256) процесорних елементів (ПЕ), що працюють за єдиною програмою, яка застосовується до вмісту власної оперативної пам'яті кожного ПЕ.

Обмін даними між процесорами здійснювався через спеціальну матрицю комунікаційних каналів. Вказана особливість комунікаційної системи дала назву "Матричні суперкомп'ютери" відповідному класу МОС.

Відзначимо, що ширший клас МОС з розподіленою пам'яттю і з довільною комунікаційною системою отримав надалі назву “Багатопроцесорні системи з масовим паралелізмом”, або МОС з МРР-архітектурою (МРР – Massively Parallel Processing). При цьому, як правило, кожен з ПЕ МРР системи є універсальним процесором, що діє за своєю власною програмою (на відміну від спільної програми для всіх ПЕ матричної МВС).

Перші матричні МОС випускалися буквально поштучно, тому їх вартість була фантастично високою. Серійні ж зразки подібних систем, такі як ICL DAP, що включали до 8192 ПЕ, з'явилися значно пізніше, проте не набули широкого поширення, зважаючи на складність програмування МОС з одним потоком управління (з однією програмою, спільною для всіх ПЕ).

Перші промислові зразки мультипроцесорних систем з'явилися на базі векторно-конвеєрних комп'ютерів в середині 80-х років.

Найбільш поширеними МОС такого типу були суперкомп'ютери фірми Cray. Проте такі системи були надзвичайно дорогими і вироблялися невеликими серіями. Як правило, в подібних комп'ютерах об'єднувалося від 2 до 16 процесорів, які мали рівноправний (симетричний) доступ до спільної оперативної пам'яті. У зв'язку з цим вони отримали назву симетричні мультипроцесорні системи (Symmetric Multi-Processing – SMP).

Як альтернатива таким дорогим мультипроцесорним системам на базі векторно-конвеєрних процесорів була запропонована ідея будувати еквівалентні за потужністю багатопроцесорні системи з великої кількості дешевих мікропроцесорів, що серійно випускалися.

Проте дуже скоро виявилося, що SMP архітектура має досить обмежені можливості з нарощування кількості процесорів в системі через різке збільшення числа конфліктів при зверненні до спільної шини пам'яті.

У зв'язку з цим виправданою представлялася ідея забезпечити кожен процесор власною оперативною пам'яттю, перетворюючи комп'ютер на об'єднання незалежних обчислювальних вузлів.

Такий підхід значно збільшив міру масштабованості багатопроцесорних систем, але у свою чергу почав вимагати розробки спеціального способу обміну даними між обчислювальними вузлами, що реалізовується зазвичай у вигляді механізму передачі повідомлень (Message Passing).

Комп'ютери з такою архітектурою є найбільш яскравими представниками MPP систем. В даний час ці два напрямки (або якісь їх комбінації) є домінуючими в розвитку суперкомп'ютерних технологій.

Щось середнє між SMPі MPP представляють NUMA-архітектури (Non Uniform Memory Access), в яких пам'ять фізично розділена, але логічно загальнодоступна. При цьому час доступу до різних блоків пам'яті стає неоднаковим. В одній з перших систем цього типу Cray T3D час доступу до пам'яті іншого процесора був в 6 разів більше, ніж до своєї власної.

На даний час розвиток суперкомп'ютерних технологій йде чотирма основними напрямками:

- векторно-конвеєрні суперкомп'ютери;
- SMP системи;
- MPP системи;
- кластерні системи.

Розглянемо основні особливості перерахованих архітектур.

**Векторно-конвеєрні суперкомп'ютери**

Перший векторно-конвеєрний комп'ютер Cray-1 з'явився в 1976 році. Архітектура його виявилася настільки вдалою, що він поклав початок цілому сімейству комп'ютерів.

Назву цьому сімейству комп'ютерів дали два принципи, закладені в архітектурі процесорів:

- конвеєрна організація обробки потоку команд;

- введення в систему команд набору векторних операцій, які дозволяють оперувати з цілими масивами даних.

Довжина векторів, що обробляються одночасно, в сучасних векторних комп'ютерах складає, як правило, 128 або 256 елементів. Очевидно, що векторні процесори повинні мати набагато складнішу структуру і по суті справи містити безліч арифметичних пристроїв.

Основне призначення векторних операцій полягає в розпаралелюванні виконання операторів циклу, в яких в основному і зосереджена велика частина обчислювальної роботи. Для цього цикли піддаються процедурі векторизації з тим, щоб вони могли реалізовуватися з використанням векторних команд.

Як правило, це виконується автоматично компіляторами під час виготовлення ними виконливого коду програми. Тому векторно-конвеєрні комп'ютери не вимагали якоїсь спеціальної технології програмування, що і виявилось вирішальним чинником їх успіху на комп'ютерному ринку. Проте, було потрібне дотримання деяких правил при написанні циклів з тим, щоб компілятор міг їх ефективно векторизувати.

Історично це були перші комп'ютери, до яких повною мірою можна було застосувати поняття суперкомп'ютер. Як правило, декілька векторно-конвеєрних процесорів (2-16) працюють в режимі зі спільною пам'яттю (SMP), утворюючи обчислювальний вузол, а декілька таких вузлів об'єднуються за допомогою комутаторів, утворюючи або NUMA, або MPP систему.

Типовими представниками такої архітектури є комп'ютери CRAY J90/t90, CRAY Sv1, NEC Sx-4/sx-5.

Рівень розвитку мікроелектронних технологій не дозволяє в даний час виробляти однокристальні векторні процесори, тому ці системи досить громіздкі і надзвичайно дорогі.

У зв'язку з цим, починаючи з середини 90-х років, коли з'явилися досить потужні суперскалярні мікропроцесори, інтерес до цього напрямку був в значній мірі ослаблений. Суперкомп'ютери з векторно-конвеєрною архітектурою стали програвати системам з масовим паралелізмом.

Проте в березні 2002 р. корпорація NEC представила систему Earth Simulator з 5120 векторно-конвеєрними процесорами, яка в 5 разів перевищила продуктивність попереднього володаря рекорду – MPP системи ASCI White з 8192 суперскалярними мікропроцесорами. Це, звичайно ж, змусило багатьох по-новому поглянути на перспективи векторно-конвеєрних систем.

**Симетричні мультипроцесорні системи (SMP)**

Характерною рисою багатопроцесорних систем SMP архітектури є те, що всі процесори мають прямий і рівноправний доступ до будь-якої точки спільної пам'яті.

Перші системи SMP складалися з декількох однорідних процесорів і масиву спільної пам'яті, до якої процесори підключалися через спільну системну шину. Проте дуже скоро виявилося, що така архітектура непридатна для створення масштабних систем.

Перша проблема, що виникла, – велика кількість конфліктів при зверненні до спільної шини. Гостроту цієї проблеми вдалося частково зняти розділенням пам'яті на блоки, підключення до яких за допомогою комутаторів дозволило розпаралелювати звернення від різних процесорів. Проте і в такому підході неприйнятно великими здавалися накладні витрати для систем більш ніж з 32-ма процесорами.

Сучасні системи SMP архітектури складаються, як правило, з декількох однорідних мікропроцесорів, що серійно випускаються, і масиву спільної пам'яті, підключення до якої виконується або за допомогою спільної шини, або за допомогою комутатора.

Наявність спільної пам'яті значно спрощує організацію взаємодії процесорів між собою і програмування, оскільки паралельна програма працює в єдиному адресному просторі.Проте за цією простотою ховаються великі проблеми, притаманні системам цього типу. Всі вони так чи інакше пов'язані з оперативною пам'яттю. Річ у тому, що зараз навіть в однопроцесорних системах найвужчим місцем є оперативна пам'ять, швидкість роботи якої значно відстала від швидкості роботи процесора.

Для того, щоб згладити цей розрив, сучасні процесори забезпечуються швидкісною буферною пам'яттю (кеш-пам'яттю), швидкість роботи якої значно вища, ніж швидкість роботи основної пам'яті.

Очевидно, що при проектуванні багатопроцесорних систем ці проблеми ще більше загострюються. Окрім добре відомої проблеми конфліктів при зверненні до спільної шини пам'яті виникла і нова проблема, пов'язана з ієрархічною структурою організації пам'яті сучасних комп'ютерів.

У багатопроцесорних системах, побудованих на базі мікропроцесорів із вбудованою кеш-пам'яттю, порушується принцип рівноправного доступу до будь-якої точки пам'яті. Дані, що знаходяться в кеш-пам'яті деякого процесора, недоступні для інших процесорів. Це означає, що після кожної модифікації копії деякої змінної, що знаходиться в кеш-пам'яті якого-небудь

процесора, необхідно виконувати синхронну модифікацію цієї змінної, розташованої в основній пам'яті.

З більшим або меншим успіхом ці проблеми вирішуються в рамках загальноприйнятої в даний час архітектури ccNUMA (cache coherent Non Uniform Memory Access). В цій архітектурі пам'ять фізично розподілена, але логічно загальнодоступна.

Це, з одного боку, дозволяє працювати з єдиним адресним простором, а, з іншого, збільшує масштабованість систем. Когерентність кеш-пам'яті підтримується на апаратному рівні, що, проте, не звільняє від накладних витрат на її підтримку.

На відміну від класичних систем SMP пам'ять стає трирівневою:

- кеш-пам'ять процесора;
- локальна оперативна пам'ять;
- віддалена оперативна пам'ять;

Час звернення до різних рівнів може відрізнятися на порядок, що сильно ускладнює написання ефективних програм для таких систем. Перераховані обставини значно обмежують можливості з нарощування продуктивності ccNUMA систем шляхом простого збільшення кількості процесорів.

Неприємною властивістю SMP систем є те, що їх вартість зростає швидше, ніж продуктивність при збільшенні кількості процесорів в системі. Крім того, через затримки під час звернення до загальної пам'яті неминуче взаємне гальмування при паралельному виконанні навіть незалежних програм.

**Системи з масовим паралелізмом (MPP)**

Проблеми, притаманні багатопроцесорним системам зі спільною пам'яттю, простим і природним чином усуваються в системах з масовим паралелізмом. Комп'ютери цього типу є багатопроцесорними системами з розподіленою пам'яттю, в яких за допомогою деякого комунікаційного середовища об'єднуються однорідні обчислювальні вузли.

Кожен з вузлів складається з одного або декількох процесорів, власної оперативної пам'яті, комунікаційного обладнання, підсистеми введення/виведення, тобто має все необхідне для незалежного функціонування.

При цьому на кожному вузлі може функціонувати або повноцінна операційна система (як в системі RS/6000 SP2), або урізаний варіант, що

підтримує лише базові функції ядра, а повноцінна ОС працює на спеціальному керівному комп'ютері (як в системах Cray T3E, nCUBE2).

Процесори в таких системах мають прямий доступ лише до своєї локальної пам'яті. Доступ до пам'яті інших вузлів реалізується зазвичай за допомогою механізму передачі повідомлень. Така архітектура обчислювальної системи усуває одночасно як проблему конфліктів при зверненні до пам'яті, так і проблему когерентності кеш-пам'яті.

Це дає можливість практично необмеженого нарощування кількості процесорів в системі, збільшуючи тим самим її продуктивність. Успішно функціонують MPP з сотням і тисячами процесорів (ASCI White – 8192, Blue Mountain – 6144).

Важливою властивістю MPP систем є їх висока міра масштабованості. Залежно від обчислювальних потреб для досягнення необхідної продуктивності потрібно просто зібрати систему з потрібною кількістю вузлів.

На практиці все, звичайно, набагато складніше. Усунення одних проблем, як це зазвичай буває, породжує інші. Для MPP систем на перший план виходить проблема ефективності комунікаційного середовища.

Різні виробники MPP систем використовували різні топології. У комп'ютерах Intel Paragon процесори утворювали прямокутну двовимірну сітку. Для цього в кожному вузлі досить чотирьох комунікаційних каналів.

У комп'ютерах Cray T3D/T3E використовувалася топологія тривимірного тора. Відповідно, у вузлах цього комп'ютера було шість комунікаційних каналів. Фірма nCUBE використовувала в своїх комп'ютерах топологію n-вимірного гіперкуба.

Кожна з розглянутих топологій має свої переваги і недоліки.

Відмітимо, що при обміні даними між процесорами, які не є найближчими сусідами, відбувається трансляція даних через проміжні вузли. Очевидно, що у вузлах мають бути передбачені якісь апаратні засоби, які звільняли б центральний процесор від участі в трансляції даних.

Останнім часом для з'єднання обчислювальних вузлів частіше використовується ієрархічна система високошвидкісних комутаторів, як це вперше було реалізовано в комп'ютерах IBM SP2. Така топологія дає можливість прямого обміну даними між будь-якими вузлами, без участі в цьому проміжних вузлів.

Системи з розподіленою пам'яттю ідеально підходять для пареллельного виконання незалежних програм, оскільки при цьому кожна програма виконується на своєму вузлі і жодним чином не впливає на виконання інших програм. Проте при розробці паралельних програм доводиться враховувати складнішу, ніж в SMP системах, організацію пам'яті.

Оперативна пам'ять в MPP системах має 3-х рівневу структуру:

- кеш-пам'ять процесорів;
- локальна оперативна пам'ять;
- оперативна пам'ять інших вузлів.

При цьому відсутня можливість прямого доступу до даних, розташованих в інших вузлах. Для їх використання ці дані мають бути заздалегідь передані в той вузол, який в даний момент їх потребує. Це значно ускладнює програмування.

Крім того, обмін даними між вузлами виконується значно повільніше, ніж обробка даних в локальній оперативній пам'яті вузлів. Тому написання ефективних паралельних програм для таких комп'ютерів є складнішим завданням, ніж для SMP систем.

**Кластерні системи**

Кластерні технології стали логічним продовженням розвитку ідей, закладених в архітектурі MPP систем.

Якщо процесорний модуль в MPP системі є закінченою обчислювальною системою, то наступний крок напрошується сам собою: чому б в якості обчислювальних вузлів не використовувати звичайні комп'ютери, що серійно випускаються.

Розвиток комунікаційних технологій, а саме, поява високошвидкісного мережевого обладнання і спеціального програмного забезпечення, такого як система MPI, що реалізовує механізм передачі повідомлень над стандартними мережевими протоколами, зробили кластерні технології загальнодоступними.

Сьогодні не складає великих труднощів створити невелику кластерну систему, об'єднавши обчислювальні потужності комп'ютерів окремої лабораторії або учбового класу. Привабливою рисою кластерних технологій є те, що вони дозволяють для досягнення необхідної продуктивності об'єднувати в єдині обчислювальні системи комп'ютери різного типу, починаючи від персональних комп'ютерів і закінчуючи потужними суперкомп'ютерами.

Широкого поширення кластерні технології набули як засіб створення систем суперкомп'ютерного класу із складових частин масового виробництва, що значно знижує вартість обчислювальної системи.

Зокрема, одним з перших був реалізований проект COCOA, в якому на базі 25 двопроцесорних персональних комп'ютерів загальною вартістю порядку $100000 була створена система з продуктивністю, еквівалентною 48-процесорному Cray T3d вартістю декілька мільйонів доларів США.

Звичайно, про повну еквівалентність цих систем говорити не доводиться. Як вказувалося, продуктивність системи з розподіленою пам'яттю дуже сильно залежить від продуктивності комунікаційного середовища.

Комунікаційне середовище можна досить повно охарактеризувати двома параметрами: латентністю – час затримки при відсиланні повідомлення та пропускною здатністю – швидкість передачі інформації. Так ось для комп'ютера Cray T3D ці параметри складають відповідно 1 мкс і 480 мб/с, а для кластера, у якому в якості комунікаційного середовища використана мережа Fast Ethernet, 100 мкс і 10 мб/с.

Це частково пояснює дуже високу вартість суперкомп'ютера. При таких параметрах, як в розглянутого кластера, знайдуться не так багато задач, які можна ефективно вирішувати на досить великій кількості процесорів.

Якщо говорити коротко, то кластер – це зв'язаний набір повноцінних комп'ютерів, що використовується як єдиний обчислювальний ресурс.

Переваги кластерної системи перед набором незалежних комп'ютерів очевидні.

По-перше, розроблено безліч диспетчерських систем пакетної обробки завдань, що дозволяють відправити завдання на обробку кластеру вцілому, а не якомусь окремому комп'ютеру. Ці диспетчерські системи автоматично розподіляють завдання за вільними обчислювальними вузлами або буферизують їх за відсутності таких, що дозволяє забезпечити більш рівномірне і ефективне завантаження комп'ютерів.

По-друге, з'являється можливість спільного використання обчислювальних ресурсів декількох комп'ютерів для вирішення одного завдання. Для створення кластерів зазвичай використовуються або прості однопроцесорні персональні комп'ютери, або двох- чи чотирипроцесорні SMP-сервери.

При цьому не накладається жодних обмежень на склад і архітектуру вузлів. Кожен з вузлів може функціонувати під управлінням своєї власної

операційної системи. Найчастіше використовуються стандартні ОС: Linux, FreeBSD, Solaris, Tru64 Unix, Windows.

У тих випадках, коли вузли кластера неоднорідні, то говорять про гетерогенні кластери.

При створенні кластеру можна виділити два підходи. Перший підхід застосовується при створенні невеликої кластерної системи. У кластер об'єднуються повнофункціональні комп'ютери, які продовжують працювати і як самостійна одиниця, наприклад, комп'ютери учбового класу або робочі станції лабораторії.

Другий підхід застосовується в тих випадках, коли цілеспрямовано створюється потужний обчислювальний ресурс. Тоді системні блоки комп'ютерів компактно розміщуються в спеціальний стійках, а для управління системою і для запуску задач виділяється один або декілька повнофункціональних комп'ютерів, так званих хост-комп’ютерів.

У цьому випадку немає необхідності забезпечувати комп'ютери обчислювальних вузлів графічними картами, моніторами, дисковими накопичувачами та іншим периферійним обладнанням, що значно знижує вартість системи.

## 4. Паралельні комп'ютери з загальною пам'яттю.

Вся оперативна пам'ять таких комп'ютерів розподіляється між кількома однаковими процесорами. Це знімає проблеми попереднього класу, але додає нові - число процесорів, що мають доступ до загальної пам'яті, з технічних причин не можна зробити великим. До даного напрямку входять багато сучасних багатопроцесорних SMP-комп'ютерів, сервер HP T600 або Sun Ultra Enterprise 5000.

Останній напрямок не є самостійним, а скоріше являє собою комбінації попередніх трьох. З кількох процесорів, традиційних або векторно-конвеєрних і загальної для них пам'яті формується обчислювальний вузол. Якщо обчислювальної потужності отриманого вузла не досить, то об'єднують кілька вузлів високошвидкісними каналами. Подібну архітектуру називають кластерною, по такому принципу побудовані CRAY SV1, HP Exemplar, Sun StarFire, NEC SX-5, останні моделі IBM SP2 і інші. Саме цей напрямок на сьогодні є найбільш перспективним.

Продуктивність багатопроцесорних систем не зростає пропорційно числу використовуваних процесорів. Відповідно принципу Амдала максимальний виграш, що можна отримати, незалежність від кількості використовуваних процесорів у системі і не перевищуватиме 10-кратного прискорення виконання програми. 10 - це теоретична верхня оцінка найкращого випадку, коли ніяких інших негативних факторів немає.

## 5. Концепція GRID та метакомп’ютінг

## (Завдання для Лабораторної роботи №1)

Ніяка обчислювальна система не може зрівнятися ні за пікової продуктивності, ні за обсягом оперативної або дискової пам'яті з тими сумарними ресурсами, якими володіють комп'ютери, підключені до Інтернету. Комп'ютер, що складається з комп'ютерів, свого роду метакомп’ютер. Звідси походить і спеціальну назву для процесу організації обчислень на такий обчислювальній системі – метакомп’ютінг . В принципі, абсолютно не обов'язково розглядати саме Інтернет в якості комунікаційного середовища метакомп’ютера, цю роль може виконувати будь-яка мережна технологія. У даному випадку для нас важливий принцип, а можливостей для технічної реалізації зараз існує достатньо. Разом з тим, до Інтернету завжди був і буде особливий інтерес, оскільки ніяка окрема обчислювальна система не зрівняється за своєю потужністю з потенційними можливостями глобальної мережі .

Конструктивні ідеї використання розподілених обчислювальних ресурсів для вирішення складних завдань з'явилися відносно недавно. Перші прототипи реальних систем метакомп'ютингу стали доступними з середини 90-х років минулого століття. Деякі системи претендували на універсальність, частина з них була орієнтована на вирішення конкретних завдань, частина на використання виділених високопродуктивних мереж і спеціальних протоколів, а десь за основу бралися звичайні канали і робота по протоколу HTTP. Ось лише кілька прикладів.

Розширення MPI для підтримки розподілених обчислень PACX - MPI. Підтримується об'єднання в єдиний метакомпьютер декількох MPP-систем, можливо з різними реалізаціями MPI . Передача даних між MPP проводиться через Інтернет за допомогою TCP / IP. На конференції Supercomputing в 1998 році продемонстровано спільне використання засобами PACX - MPI двох 512 - процесорних суперкомп'ютерів Cray T3E , що знаходяться в університеті Штутгарта ( Німеччина ) і в Пітсбурзькому суперкомпьютерном центрі.

GIMPS - Great Internet Mersenne Prime Search . Пошук простих чисел Мерсена , тобто простих чисел виду $2^P$— 1, де P є простим числом. У листопаді 2001 року в рамках даного проекту було знайдено максимальне на даний час число Мерсена $2^{13\ 466\ 917}$ — 1. Десятки тисяч комп'ютерів по всьому світу , віддаючи частину своїх обчислювальних ресурсів , працювали над цим завданням два з половиною роки. Організація Electronic Frontier Foundation пропонує приз в $ 100 000 за знаходження простого числа Мерсена , що містить 10 мільйонів цифр. Адреса проекту http://mersenne.org/.

Проект Globus спочатку зародився в Аргонської національної лабораторії і зараз отримав широке визнання в усьому світі. Метою проекту є створення засобів для організації глобальної інформаційно-обчислювального середовища. У рамках проекту розроблено цілий ряд програмних засобів і систем, зокрема , однаковий інтерфейс до різних локальних систем розподілу навантаження, системи аутентифікації, комунікаційна бібліотека Nexus, засоби контролю і моніторингу та інші. Розроблені засоби поширюються вільно у вигляді пакету Globus Toolkit разом з вихідними текстами. В даний час Globus взятий за основу в безлічі інших масштабних проектів, таких як National Technology Grid, Information Power Grid і European DataGrid. Додаткову інформацію можна знайти на сайті http://www.globus.org/ .

Прогрес в мережевих технологіях останніх років колосальний. Гигабітні лінії зв'язку між комп'ютерами, рознесеними на сотні кілометрів, стають звичайною реальністю. Об'єднавши різні обчислювальні системи в рамках єдиної мережі, можна сформувати спеціальну обчислювальну середу. Якісь комп'ютери можуть підключатися або відключатися, але, з точки зору користувача, ця віртуальна середовище є єдиним метакомпьютером. Працюючи в такому середовищі, користувач лише видає завдання на

вирішення завдання, а решта метакомпьютер робить сам: шукає доступні обчислювальні ресурси, відстежує їх працездатність, здійснює передачу даних, якщо потрібно, то виконує перетворення даних у формат комп'ютера, на якому виконуватиметься завдання, і т. п. Користувач може навіть і не дізнатися, ресурси якого саме комп'ютера були йому надані. А, за великим рахунком, чи часто вам це потрібно знати? Якщо потрібні обчислювальні потужності для вирішення завдання, то ви підключаєтеся до метакомп’ютеру, видаєте завдання і отримуєте результат. Все.

Тут існує майже повна аналогія з електричною мережею. Підключаючи електричний чайник до розетки, ви не замислюєтеся, яка станція виробляє електроенергію. Вам потрібен ресурс, ви їм користуєтеся. До речі, за аналогією саме з електричною мережею розподілена обчислювальна середу в англомовній літературі отримала назву Grid або "обчислювальна мережа". Надалі, слова Grid і метакомп’ютер ми будемо використовувати як синоніми.

Продовжуючи аналогію з електричною мережею, на метакомп’ютер хотілося б перенести не тільки назва, а й такий же простий спосіб взаємодії з ним користувача. Але ось тут і виникають основні проблеми, які визначаються складністю організації самого метакомп’ютера . На відміну від традиційного комп'ютера метакомп’ютер має цілий набір властивих тільки йому особливостей:

- метакомпьютер володіє величезними ресурсами, які непорівнянні з ресурсами звичайних комп'ютерів. Це стосується практично всіх параметрів: число доступних процесорів, обсяг пам'яті, число активних додатків, користувачів і т. п.;
- метакомпьютер є розподіленим за своєю природою. Компоненти метакомпьютера можуть бути віддалені один від одного на сотні й тисячі кілометрів , що неминуче викличе велику латентність і, отже, позначиться на оперативності їх взаємодії;
- метакомпьютер може динамічно змінювати конфігурацію. Якісь комп'ютери до нього приєднуються і делегують права на використання своїх ресурсів, якісь відключаються і стають недоступними. Але для користувача робота повинна бути прозорою. Завдання системи підтримки роботи метакомпьютера складається в пошуку відповідних ресурсів, перевірці їх працездатності, у розподілі вступників завдань незалежно від поточної конфігурації метакомпьютера в цілому;
- метакомпьютер неоднорідний. При розподілі завдань потрібно враховувати особливості операційних систем, що входять до його складу. Різні системи підтримують різні системи команд і формати представлення даних. Різні системи в різний час можуть мати різну завантаження, зв'язок з обчислювальними системами йде по каналах з різною пропускною здатністю. Нарешті, до складу метакомпьютера можуть входити системи

з принципово різною архітектурою , починаючи з домашніх персональних комп'ютерів, закінчуючи найпотужнішими системами зі списку Top500;

- метакомпьютер об'єднує ресурси різних організацій. Політика доступу та використання конкретних ресурсів може сильно змінюватися в залежності від їх приналежності до тієї чи іншої організації. Метакомп'ютер не належить нікому, тому політика його адміністрування може бути визначена лише в найзагальніших рисах. Разом з тим, узгодженість роботи величезної кількості складових частин метакомпьютера передбачає обов'язкову стандартизацію роботи всіх його служб і сервісів.

В даний час йде активне обговорення різних стратегій побудови метакомпьютера. Однак багато питань до цих пір недостатньо опрацьовані, частина запропонованих технологій ще знаходиться на стадії апробації, не завжди використовується єдина термінологія. Ситуація в даній області розвивається надзвичайно швидко. У даній книзі ми вирішили основну увагу приділити не детальному опису будь-яких конкретних систем або технологій метакомп'ютингу, а обговоренню базових принципів та ідей обчислювальної компоненти мережі. Багато корисної інформації про поточний стан робіт у цій галузі можна знайти на сайтах http://www.globus.org, http://www.gridforum.org та інші.

Незважаючи на гадану химерність і нереальність створюваної глобальної обчислювальної системи, область застосування метакомпьютера обширна.

У даному ряду варто особливо відзначити задачі розподіленого зберігання і обробки даних. Реально масиви даних можуть бути географічно віддалені один від одного і розподілені по великому числу різного роду сховищ і баз даних. Незважаючи на роз'єднаність, всі ці дані можуть знадобитися в рамках єдиного експерименту, що пред'являє до метакомпьютеру серйозні вимоги не тільки обчислювального, а й комунікаційного характеру.

Реальна робота по створенню та апробації систем метакомп'ютингу активно йде з багатьох напрямків. Багато провідні компанії взяли Globus Toolkit в якості стандарту для створення Grid-додатків, створюючи програмну інфраструктуру для своїх платформ. Створюються глобальні полігони, об'єднуючі в рамках супершвидкісних мереж значні розподілені обчислювальні ресурси. Проводяться серії експериментів, спрямованих на відпрацювання нових мережевих технологій, методів диспетчеризації та моніторингу в розподіленої обчислювальної середовищі, інтерфейсу з користувачем, моделей і методів програмування. Потенціал напрямки, безумовно, величезний, але число невирішених проблем поки переважує

реальний ефект. Зараз все знаходиться в стадії становлення. Немає сумнівів, що в майбутньому ситуація зміниться.

**Завдання для лабораторної роботи №1**.

1) Напишіть тестову програму , яка автоматично визначає співвідношення між швидкістю виконання арифметичних операцій і швидкістю обміну з різними рівнями пам'яті.
2) Напишіть тестову програму , яка автоматично визначає різницю в часі звернення до локальної та віддаленої пам'яті для обчислювальних систем з архітектурою ccNUMA .
3) Напишіть програму, що працює з максимальною продуктивністю на будь-якій доступній вам обчислювальної системі із загальною пам'яттю. Побудуйте залежність продуктивності від числа процесорів.
4) Зробіть Web- інтерфейс до своєї паралельній програмі з можливістю завдання вхідних параметрів і запуску програми на різних конфігураціях доступного паралельного комп'ютера.

## 6. Методи оцінки продуктивності паралельних алгоритмів і систем (Завдання для Лабораторної роботи №2)

**1. Загальні зауваження стосовно оцінки продуктивності паралельних систем.**

**2. Фактори, що необхідно враховувати при оцінці продуктивності.**

**3. Методи оцінки продуктивності паралельних систем.**

**4. Характеристики продуктивності паралельних алгоритмів.**

**5. Порівняння MIMD і SIMD структур за продуктивністю.**

### 1. Загальні зауваження стосовно оцінки продуктивності паралельних систем

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

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

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

В процесі пошуку стандартної одиниці вимірювання продуктивності комп’ютерів було прийнято декілька популярних одиниць вимірювання, а

для оцінки продуктивності паралельних вузлів деякі терміни були штучно вирвані з їх контексту і використані там, для чого вони ніколи не призначалися. Насправді єдиною і надійною одиницею вимірювання продуктивності є час виконання реальних програм, і всі пропоновані заміни цього часу як одиниці вимірювання або заміни реальних програм як об'єктів вимірювання на синтетичні програми тільки вводять в оману.

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

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

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

Приклад для останнього випадку.

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

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

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

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

Проте і з використанням MFLOPS для оцінки продуктивності виникають певні проблеми. Перш за все, це пов'язане з тим, що *набори операцій з рухомою крапкою не сумісні на різних комп'ютерах.* Наприклад, в суперкомп'ютерах фірми Cray Research відсутня команда ділення (є, правда, операція обчислення оберненої величини числа з рухомою крапкою, а

операція ділення може бути реалізована за допомогою множення діленого на зворотну величину дільника). В той же час сучасні мікропроцесори мають команди ділення, обчислення квадратного кореня, синуса і косинуса, тощо.

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

Для усунення цих недоліків *використовується "нормалізоване" число операцій з рухомою крапкою,* за тестовим пакетом "Ліверморські цикли" (табл.2.1.).

Таблиця 2.1.Співвідношення між реальними і нормалізованими операціями з рухомою крапкою.

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

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

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

## 2.Фактори, що необхідно враховувати при оцінці продуктивності

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

*Тип алгоритму.* Алгоритм, що ідеально пристосований для роботи на одній архітектурі, на іншій (з цією ж кількістю процесорів) може працювати

набагато гірше. Для масивно-паралельних систем необхідний масштабований алгоритм (для "оптимального" завантаження всіх процесорів). Виграш дає оптимальне поєднання "структура - алгоритм". Тобто, *на паралельній машині продуктивність залежить від відповідності між структурою апаратних елементів і структурою обчислень в алгоритмі. Не можна переносити результат з однієї на іншу систему.*

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

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

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

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

### 3.Методи оцінки продуктивності паралельних систем.

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

*Метод обчислення продуктивності складових частин.* Для паралельних систем неефективний.

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

Використовується *дві групи тестів:*

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

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

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

## 4.Характеристики продуктивності паралельних алгоритмів.

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

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

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

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

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

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

- $T_1$ – час виконання алгоритму на одному процесорі чи однопроцесорній системі;

- $T_N$ – час виконання алгоритму на багатопроцесорній системі з N - процесорами.

*Закон Амдала.* Позначимо:

- $P_c$- максимальний ступінь розпаралелення (максимальна кількість процесорів, які можуть працювати паралельно в будь-який момент часу в період виконання програми). Тут вирішальне значення має також тип застосовуваної моделі паралельності (наприклад, MIMD чи SIMD);

- $T_k$- тривалість виконання програм з максимальним показником розпаралелення $P_c \geq k$ на системі з k процесорами;

- $N$ - кількість процесорів у паралельній системі;

- $f$ – відсоток послідовних операцій програми (не можуть бути виконані паралельно на $N$ процесорах).

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

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

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

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

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

$$1 \leq S_N \leq N,$$

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

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

$$E_N = S_N / N$$

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

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

Приклади застосування закону Амдала

1. Система має N=1000 процесорів

• Програма має максимальний показник розпаралелення 1000

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

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

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

2. Система має N=1000 процесорів

• Програма має максимальний показник розпаралелювання 1000.

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

Показник прискорення:

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

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

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

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

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

На рис.2.1 наведені графіки залежності показника прискорення від кількості процесорів *N* для різних величин послідовної частини програми f. Головна діагональ належить до лінійного прискорення і може бути побудована при f=0.

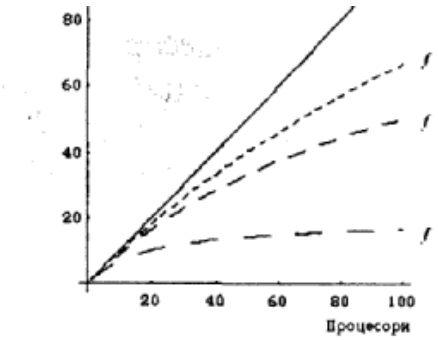



**Рис.** 2.1. Залежність прискорення від кількості процесорів

На рис. 2.2 наведені графіки залежності ефективності від *N* і f.

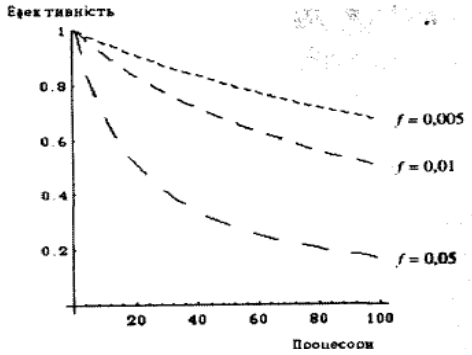



Рис. 2.2. Залежність ефективності від кількості процесорів.

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

*Ціна* (робота) обчислення визначається як *Cost = (час виконання) * (повне число використаних процесорів)*. Ціна послідовних обчислень рівна часу їх виконання $T_1$. Ціна паралельних обчислень рівна

$T_N$*$N$**.** З іншого боку $T_N = T_1/S_N$. Звідси ціна паралельних обчислень: $Cost = T_1*N/ S_N = T_1/ E$ **.** Ціноптимальні паралельні алгоритми **–** алгоритми, в яких ціна для вирішення проблеми на багатопроцесорній системі є пропорційною ціні (часу виконання) на однопроцесорній системі.

*Масштабність (Scaleup)*

Оскільки для вимірювання ефективності застосовується та сама програма, то показник $f$ в законі Амдала залишається незмінним. Проте, кожна паралельна програма, незалежно від її величини, має послідовно обробляти деяку постійну мінімальну кількість $f$ своїх інструкцій. Це означає, що графіки, які наведені на рис.2.1 і рис. 2.2 із зміною розмірів проблеми стають недійсними! Практичні заміри на паралельних системах з дуже великою кількістю процесорів показали: чим більшими є вирішувана проблема і кількість використовуваних процесорів, тим меншою стає відсоток послідовної частини обчислень. Це дає підставу зробити висновок, що на паралельній системі з дуже великою кількістю процесорів можна досягти високої ефективності.

*Повний час виконання паралельного алгоритму* $t_p$ є сумою часу, що витрачається на обчислення $t_{comp}$, і часу, що витрачається на комунікації (обмін даними між процесорами) $t_{comm}$. Час обчислень оцінюється як час для послідовного алгоритму. $t_p=t_{comp}+t_{comm.}$ Позначимо час запуску (startup), що інколи називають часом скритого стану повідомлень (message latency), як $t_{startup}$. Як початкову апроксимацію *часу комунікації* візьмемо: $t_{comm}=t_{startup}+n * t_{data}$. Будемо рахувати $t_{startup}$ постійним і залежним як від обладнання, так і від програмного забезпечення. Час передачі одного слова даних $t_{data}$ також вважатимемо постійним. Нехай від одного до другого процесора передаються $n$ даних, *тоді теоретичний час комунікацій* можна представити в виді графіка (див.рис.2.3).

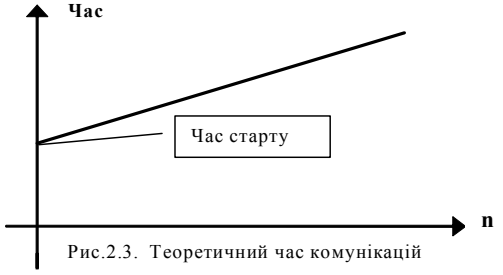



Рис.2.3. Теоретичний час комунікацій

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

***Визначення для варіантів програми, що мають різну величину***

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

Показник масштабованості деякої проблеми, величина якої n на k процесорах порівняно з меншою проблемою *m* (m<n) на одному процесорі визначається так: якщо $T_1(m) = T_k(n)$, тобто тривалість виконання "малої" програми на одному процесорі дорівнює тривалості виконання "великої" програми на k процесорах, то показник масштабності можна виразити формулою:

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

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

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

Графік залежності показника збільшення складності вирішуваних задач SCк від кількості процесорів наведений на рис.2.4.

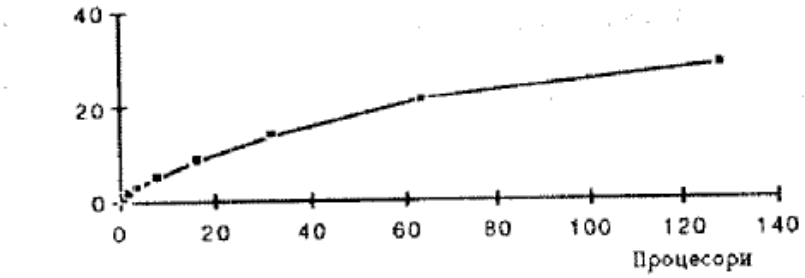



*Рис.2.4. Залежність показника збільшення складності вирішуваних задач SCк від кількості процесорів*

**5. Порівняння MIMD і SIMD структур за продуктивністю.**

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

На практиці необхідно звернути увагу на такі моменти:

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

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

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

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

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

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

- $f_A$ - послідовну частину програми відносно кількості елементарних операцій;

- $f_B$ - послідовну частину програми відносно тривалості виконання операцій.

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

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

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

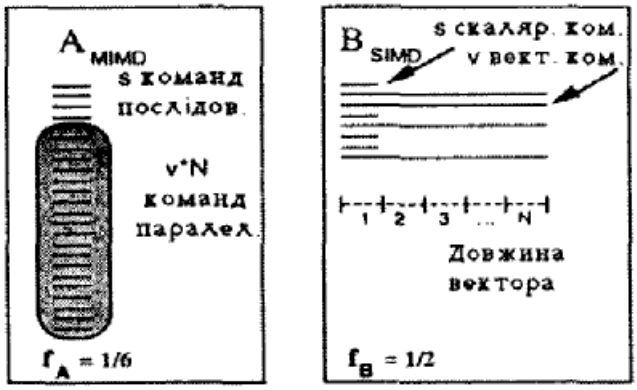



***Рис. 2.5*** *Оцінка кількості паралельних інструкції*

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

*Визначення:*

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

Можна вважати ідентичним і таке визначення

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

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

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

Приклади застосування зміненого закону

1.

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

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

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

Приблизний результат одержуємо і за формулою Амдала з відповідним $f_A$ .

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

Показник прискорення за формулою Амдала:

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

2.

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

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

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

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

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

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

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

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

Крім того, завжди виконується умова:

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

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

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

*Висновки*

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

1. Характеристики продуктивності, як показник прискорення і показник масштабності завжди пов'язані з конкретним застосуванням паралельних систем - показники справедливі тільки для даного класу задач і дуже умовно можуть бути поширені на інші задач.
2. Показник прискорення деякої програми стосується тільки одного окремого процесора паралельної системи.

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

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

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

## 7. Сучасна технологія паралельного програмування: використання традиційних послідовних мов, систем програмування на основі передачі повідомлень, Т-система, система НОРМА. (Завдання для Лабораторної роботи №3)

Широке поширення комп'ютерів з розподіленою пам'яттю визначило і поява відповідних систем програмування. Як правило, в таких системах відсутня єдиний адресний простір, і для обміну даними між паралельними процесами використовується явна передача повідомлень через комунікаційне середовище. Окремі процеси описуються за допомогою традиційних мов програмування , а для організації їх взаємодії вводяться додаткові функції. З цієї причини практично всі системи програмування , засновані на явною передачі повідомлень, існують не у вигляді нових мов, а у вигляді інтерфейсів і бібліотек.

До теперішнього часу прикладів відомих систем програмування на основі передачі повідомлень накопичилося досить багато: Shmem, Linda, PVM , MPI та інші.

### Т -система

У цьому розділі ми розглянемо ключові аспекти технології автоматичного динамічного розпаралелювання програм, відомої як Т-система. Розробка Т-системи була почата в Інституті програмних систем РАН (м. Переславль -Залеський) наприкінці 80-х років минулого сторіччя.

Найбільш характерною рисою Т-системи є використання парадигми функціонального програмування для забезпечення динамічного розпаралелювання програм. На цій основі вдалося знайти і реалізувати в Т-системі цікаві форми для організації паралельних обчислень, зокрема, для

синхронізації або розподілу навантаження. Разом з тим, функціональний стиль Т-системи вдалося вдало поєднати з традиційними мовами програмування за допомогою розширень мов С, С + + або мови Fortran. Явні паралельні конструкції в мові відсутні, і програміст в тексті явно не вказує, які частини програми слід виконувати паралельно.

Базові принципи Т-системи спираються на результати загальної теорії функціонального програмування. Для їх пояснення можна не вдаватися в деталі теорії, а скористатися простою аналогією. Припустимо, що у нас є складне арифметичне вираз, що включає багато підвирази, укладених в дужки. Всі ці підвирази можна обчислювати в будь-якому порядку , і в кожному випадку ми отримаємо один і той же результат (звичайно ж, розглядається не машинна, а точна арифметика без помилок округлення). У теорії функціонального програмування цей закон арифметики узагальнюється на довільні рекурсивні функції.

Такий підхід дає прямий метод для розпаралелювання функціональних програм , побудованих з "чистих " функцій. Чисті функції - це одне з базових понять Т-системи, що позначає функції без побічних ефектів. У кожен момент часу необхідно виділяти готові до обчислення "підвираз" і розподіляти їх за наявними процесорам. За основу береться граф, вузли якого представляють викликані функції, а дуги відповідають відношенню" підвираз - вираз".

Виявляється, що для додавання функціональної семантики в традиційній мові програмування досить ввести поняття неготового значення . У мові С це досягається введенням додаткового атрибута у описі змінних. Нове ключове слово tvai в описі "tvai int i" визначає змінну, значення якої може бути цілим числом або неготовим значенням (поки не порахуємо). Для позначення функцій без побічних ефектів додатково використовується слово tfun, вихід Т- функції позначається словом tout і т. д.

**Система програмування НОРМА**

Мова НОРМА є спеціалізованим непроцедурною мовою, призначеною для специфікації завдань обчислювального характеру, задач математичної фізики. Всі конструкції мови носять декларативний характер і описують правила обчислення значень. Основне призначення мови полягає в автоматизації процесу розробки програм. Програміст працює "майже" в термінах математичних формул, що значно спрощує його роботу. Завдання транслятора ускладнюється. Крім традиційних завдань, синтаксичного і семантічного аналізу, він виконує синтез вихідний програми.

Ідеї були сформульовані І. Б. Задихайло в роботі ще в 1963 році. Подальший їх розвиток призвело до появи мови НОРМА і не - скількох версій транслятора для різних платформ.

Спочатку термін НОРМА розшифровувався як Непроцедурного Опис Різницевих Моделей Алгоритмів. У наслідку з'явилася й інша трактування: нормальний рівень спілкування прикладного математика з комп'ютером. Розробник прикладних програм абстрагується від особливостей конкретних комп'ютерів і мислить у звичних термінах своєї предметної області. Відштовхуючись від конкретних потреб Інституту прикладної математики ім. М.В. Келдиша РАН, автори мови намагалися максимально спростити рішення класу задач математичної фізики. Специфіка предметної області - це орієнтація на сіточні методи. Саме цей факт наклав значний відбиток як на концепцію мови, так і на всі його основні конструкції.

У записі програми на мові НОРМА не потрібно ніякої інформації про порядок виконання операцій. Порядок пропозицій мови може бути довільній. Мова дозволяє формулювати запит на обчислення, не уточнюючи, яким саме чином обчислення слід організувати. Всі інформаційні зв'язки виявляються і враховуються транслятором - синтезатором на етапах аналізу вихідної програми і синтезу вихідного тексту. На трансляторі лежить і вибір конкретного способу організації обчислень. Зокрема, на етапі синтезу результуючої програми він може згенерувати як послідовний, так і паралельний код.

Вибір високого рівня мови НОРМА визначає його характерну рису - це мова з одноразовим привласненням . Кожна змінна може приймати значення тільки один раз. Такі поняття, як пам'ять, побічний ефект, оператор присвоювання та керуючі оператори в мові НОРМА відсутні просто "за визначенням". У всіх традиційних мовах програмування ці поняття є, оскільки з їх допомогою потрібно формулювати конкретний алгоритм з урахуванням питань економії та розподілу пам'яті, порядку виконання операторів і т.п. Запис на мові НОРМА, по суті, є записом чисельного методу вирішення конкретного завдання.

Приклад програми , в якій всі процеси обмінюються повідомленнями з найближчими сусідами відповідно до топології кільця.

```
# include " mpi.h " # include <stdio.h>
int main ( argc , argv )
int argc ;
char * argv [] ; {
int numtasks , rank , next , prev , buf [ 2 ] , tagl = l , tag2 = 2 ;
MPI_Request reqs [ 4 ] ;
MPI_Status stats [ 4 ] ;
MPI_In.it ( Sargc , & argv ) ;
MPI_Comm_size ( MPI_COMM_WORLD , Snumtasks ) ;
MPI_Comm_rank ( MPI_COMM_WORLD , Srank ) ;
prev = rank - 1;
```

```
next = rank +1;
if ( rank == 0 ) prev = numtasks - 1;
if ( rank == ( numtasks - 1 )) next = 0;
MPI_Irecv (&buf [0], 1, MPI_INT, prev, tagl, MPI_COMM_WORLD,
&reqs [0]);
MPI_Irecv (&buf [1], 1, MPI_INT, next, tag2, MPI_COMM_WORLD, & reqs [1]);
MPI_Isend (& rank, 1, MPI_INT, prev, tag2, MPI_COMM_WORLD, &reqs [ 2]);
MPI_Isend (&rank, 1, MPI_INT, next, tagl, MPI_COMM_WORLD, &reqs [ 3]);
MPI_Waitall ( 4 , reqs , stats ) ;
MPI_Finalize ();
}
int MPI_Waitany ( int count , MPI_Request * requests , int * index , MPI_Status *
status )
```

- count - число ідентифікаторів асинхронних операцій;
- requests - ідентифікатори операцій асинхронного прийому або передачі;
- OUT index - номер завершеною операції обміну;
- OUT status - параметри повідомлення.

Виконання процесу блокується до тих пір, поки яка-небудь асинхронна операція обміну, асоційована з вказаними ідентифікаторами, що не буде завершена. Якщо завершилися кілька операцій, то випадковим чином буде обрана одна з них. Параметр index містить номер елемента в масиві requests, містить ідентифікатор завершеною операції.

**Завдання 1:** показати використання функції MPi_scatter для розсилки рядків масиву. Нагадаємо, що в мові С, на відміну від Fortran, масиви зберігаються в пам'яті по рядках.

**Завдання 2:** З використанням системи Linda напишіть програму, що реалізовує додавання елементів вектора за схемою здвоювання.

## 8.Стеки протоколів комп’ютерних мереж

Стек протоколів – це ієрархічно впорядкована сукупність протоколів, достатніх для реалізації взаємодії вузлів у комп’ютерній мережі.

Існує досить багато стеків протоколів, які широко використовуються у мережах. Це стеки, які з'явилися на основі міжнародних і національних стандартів та стеки, запропоновані фірмами-виробниками мережевого обладнання, які одержали поширення завдяки поширеності обладнання саме цих фірм.

Прикладами популярних стеків протоколів можуть служити: стек IPX/SPX фірми Novell, стек TCP/IP, що використовується у мережі Internet і в багатьох мережах на основі операційної системи UNIX, стек Decnet корпорації Digital Equipment і деякі інші.

Застосування в мережі різних стеків комунікаційних протоколів породжує велику різноманітність характеристик і структур цих мереж. У невеликих мережах достатньо використання одного стеку, але у великих корпоративних мережах, що поєднують різні підмережі, як правило, паралельно використовуються декілька стеків.

Протоколи можуть бути реалізовані у вигляді програмних елементів операційної системи. Наприклад, дуже часто протоколи канального рівня виконані у вигляді драйверу мережевого адаптеру, а функції протоколів верхніх рівнів представляються серверними або клієнтськими компонентами мережевих служб.

У комунікаційному обладнанні реалізуються протоколи нижніх рівнів, які більш стандартизовані, ніж протоколи верхніх рівнів, що є передумовою для успішної спільної роботи обладнання від різних виробників.

Наприклад, на фізичному та канальному рівнях практично у всіх стеках використовуються ті самі протоколи. Це добре стандартизовані протоколи Ethernet, Token Ring, FDDI та інші, що дозволяють використовувати у всіх мережах однакову апаратуру.

Протоколи більш високих рівнів, починаючи з мережевого, у існуючих стандартних стеках відрізняються більшою різноманітністю та найчастіше не відповідають рекомендованій моделлю OSI розбивці на рівні. Наприклад, функції сеансового рівня і рівня представлення можуть бути об'єднані із прикладним рівнем.

Така невідповідність пояснюється тим, що мережева модель OSI з'явилася як результат узагальнення вже існуючих і реально використовуваних стеків, а не навпаки.

**Стек протоколів OSI**

Кожному рівню моделі OSI відповідає один або кілька протоколів, які виконують функції забезпечення мережевої взаємодії.

Стек протоколів OSI відповідає моделі OSI і включає протоколи для всіх семи рівнів (табл. 1).

На фізичному і канальному рівнях стека OSI використовуються стандартні протоколи Ethernet, Token Ring тощо.

Мережевий рівень реалізований за допомогою протоколів ES-IS і IS-IS.

*ES-IS (End System to Intermediate System routing exchange protocol)* – протокол маршрутизації кінцевих систем, за допомогою якого кінцеві системи (робочі станції) сповіщають про себе проміжні системи (наприклад, концентратори).

*IS-IS (Intermediate System to Intermediate System routing exchange protocol)* – протокол маршрутизації проміжних станцій, за допомогою якого проміжні системи обмінюються інформацією про діючі маршрути в мережі.

Ці протоколи використовуються для «розвідки» і побудови повної, послідовної картини топології мережі, щоб забезпечити можливість маршрутизації пакетів, які пересилаються.

Транспортний, сеансовий і рівень представлення реалізовані відповідними протоколами OSI, які мають мале поширення.

Таблиця 1 – Стек протоколів OSI

| **Рівень моделі OSI** | **Протоколи OSI** |
|---|---|
| 7. Прикладний | FTAM, VTP, X.400 і X.500 |
| 6. Представлення | Протокол представлення OSI |
| 5. Сеансовий | Сеансовий протокол OSI |
| 4. Транспортний | Транспортний протокол OSI |
| 3. Мережевий | ES-IS, IS-IS |
| 2. Канальний | Ethernet, Token Ring, FDDI, X.25, ISDN, ATM, LAP-D, PPP та інші. |
| 1. Фізичний | Специфікації фізичних середовищ |

Найбільшу популярність отримали протоколи прикладного рівня стеку OSI. Це, передусім, протоколи FTAM, VTP, X.400 та X.500.

*FTAM (File Transfer, Access and Management)* – протокол передачі, забезпечення доступу і управління файлами. Не набув широкого поширення. Використовується німецькими банками для передачі клірингової інформації.

*VTP (Virtual Terminal Protocol)* – протокол, що описує роботу віртуального терміналу.

*X.400* –набір рекомендацій Міжнародного консультативного комітету з телеграфії та телефонії (CCITT – *від франц. Comité Consultatif International Téléphonique et Télégraphique*), у яких описуються системи пересилки

електронних повідомлень. Протокол X.400 визначає структуру повідомлень електронної пошти (e-mail) так, що всі повідомлення задовольняють стандартному формату. Використовується у таких продуктах як, наприклад, Microsoft Exchange.

*X.500* – розширення стандарту X.400, який визначає формат адреси повідомлення, що й дозволяє всім системам електронної пошти зв'язуватися між собою. З самого початку метою рекомендацій X.500 є розробка стандартів глобальної довідкової служби. Однак процес доставки повідомлення вимагає знання адреси одержувача. При великих розмірах мереж виникає проблема зберігання, пошуку й одержання адрес. Рішенням цієї проблеми є довідкова служба, яка допомагає одержувати адреси відправників і одержувачів, що й представляє собою розподілену базу даних імен і адрес.

Модель OSI зробила популярною ідею загальної моделі рівнів протоколів, яка визначає взаємодію між мережевим обладнанням і програмним забезпеченням. Проте, стек протоколів OSI, розроблений як частина проекту й спрямований забезпечити однорідність при побудові мереж, і, як наслідок, універсальність взаємодії, був сприйнятий багатьма як занадто ускладнений і мало реалізований. Справа в тому, що розробка й впровадження стеку OSI припускала відмову від існуючих протоколів і перехід на нові на всіх рівнях стеку. Це сильно ускладнило реалізацію стеку й послужило причиною для відмови від нього багатьох компаній, що зробили значні інвестиції в інші мережеві технології.

Таким чином, коли були реалізовані протоколи для моделі OSI, виявився ряд проблем:

- протоколи засновані на концепціях, які мають мало сенсу в сучасних мережах;
- специфікації, у деяких випадках, виявилися неповними або такими, що суперечать одна одній;
- за функціональними можливостями протоколи ISO/OSI поступалися іншим протоколам;
- наявність великої кількості рівнів вимагає більшої обчислювальної потужності і, як наслідок, призводить до зменшення швидкодії.

**Стек протоколів TCP/IP**

Стек TCP/IP, який також часто називається стеком Інтернет, сьогодні є найбільш популярним і таким, що швидко розвивається (табл. 2).

Таблиця 2 – Стек протоколів TCP/IP

| Рівні моделі OSI | Протоколи TCP/IP | Рівні TCP/IP |
|---|---|---|
| 7 | HTTP, FTP, TFTP, Telnet, SSH, SMTP, SNMP та інші | 1 |
| 6 | | |
| 5 | TCP, UDP | 2 |
| 4 | | |
| 3 | IP, ICMP, IGMP | 3 |
| 2 | Не регламентовано, але підтримуються всі популярні стандарти | 4 |
| 1 | | |

Цей стек був розроблений з ініціативи Міністерства оборони США й орієнтувався на забезпечення зв'язку різнорідних обчислювальних мереж.

Оскільки стек протоколів TCP/IP був розроблений до появи мережевої моделі ISO/OSI, то відповідність його рівнів рівням моделі OSI носить досить умовний характер, хоча він також має багаторівневу структуру.

Стек був реалізований для роботи в операційній системі Unix, популярність якої привела до широкого поширення протоколів TCP/IP, завдяки яким стек і одержав свою назву.

Найнижчий рівень стеку – рівень інтерфейсу з мережею відповідає фізичному й канальному рівням моделі OSI. У стеку TCP/IP цей рівень не регламентований, але реалізована підтримка практично всіх популярних стандартів фізичного й канального рівня: Ethernet, Token Ring, FDDI (для локальних мереж), X.25, ISDN, SLIP/PPP (для глобальних мереж).

Рівень міжмережевої взаємодії (рівень 3) забезпечує маршрутизацію й передачу даних мережею, виконуючи, таким чином, функції, відповідні до мережевого рівня моделі OSI. На цьому рівні використовуються протоколи IP, ICMP, IGMP.

*IP (Internet Protocol)* – міжмережевий протокол, який забезпечує передачу даних у мережах. Протокол IP специфікований в RFC 791. До його основних функцій належать адресація та фрагментація пакетів. Протокол не гарантує надійну доставку даних, не має механізму підтверджень доставки повідомлень, не виконує контроль помилок для поля даних, не підтримує повторну передачу та не виконує функцію управління потоком (flow control). Виявлені помилки можуть бути оголошені за допомогою протоколу ICMP, який підтримується модулем IP протоколу.

*ICMP (Internet Control Message Protocol)* – протокол міжмережевих керуючих повідомлень, призначений для організації зворотного зв'язку з окремими вузлами мережі при обміні інформацією про помилки, наприклад, про неможливість доставки пакету, про перевищення часу життя або

тривалості складання пакета із фрагментів, про ненормальні значення параметрів. Крім того, за допомогою цього протоколу передаються пакети, які використовуються для тестування, і пакети, які містять службові інформаційні повідомлення, наприклад, про зміну маршруту пересилання й типу обслуговування, про стан системи тощо. Протокол ICMP не робить протокол IP засобом надійної доставки повідомлень. Для цього існує TCP.

*IGMP (Internet Group Management Protocol)* – протокол, що використовується IP-вузлами і маршрутизаторами, для того щоб підтримувати групову розсилку повідомлень. Він дозволяє всім системам фізичної мережі знати, які IP-вузли в даний час об'єднані в групи і до яких груп вони належать. Ця інформація необхідна для групових маршрутизаторів, саме так вони дізнаються, які групові дейтаграми необхідно перенаправляти і на які інтерфейси. IGMP визначений в RFC 1112.

Рівень 2 стеку TCP/IP називається основним і забезпечує функції транспортування інформації з мережі. При цьому використовуються два протоколи TCP і UDP, що реалізують різні механізми, доставки даних та мають різні ступені надійності.

*TCP (Transmission Control Protocol)* – протокол керування передачею, що працює з установкою логічного з'єднання між віддаленими прикладними процесами, а також використовує принцип автоматичної повторної передачі пакетів, які містять помилки. TCP визначений в RFC 793.

*UDP (User Datagram Protocol)* – протокол користувальницьких дейтаграм (синонім терміну «пакет»), який є спрощеним варіантом TCP і працює без встановлення логічного з'єднання, відповідно, не забезпечує перевірку на наявність помилок і підтвердження доставки пакету. UDP визначений в RFC 768.

Верхній рівень стеку TCP/IP називається прикладним. До протоколів цього рівня належать такі широко використовувані протоколи, як HTTP, FTP, telnet, SMTP, SNMP і багато інших.

*HTTP (HyperText Transfer Protocol)* – протокол передачі гіпертексту. Основою HTTP є технологія «клієнт-сервер», тобто клієнти ініціюють з'єднання і посилають запит, а сервери очікують з'єднання для отримання запиту, роблять необхідні дії і повертають назад повідомлення з результатом. HTTP насьогодні використовується у Всесвітній павутині для отримання інформації з веб-сайтів.

*FTP (File Transfer Protocol)* – протокол передачі файлів, який використовує у якості транспортного протокол із встановленням з'єднань – TCP, що підвищує надійність передачі файлів. Протокол, призначений для забезпечення передачі та прийому файлів між серверами та клієнтами.

*TFTP (Trivial File Transfer Protocol)* – найпростіший протокол передачі файлів. На відміну від FTP цей протокол базується на роботі з UDP, при цьому протокол реалізує тільки передачу файлів.

*SNMP (Simple Network Management Protocol)* – простий протокол керування мережею, призначений для передачі інформації, що визначає формати повідомлень, якими обмінюються клієнти й сервери, а також формати імен і адрес вузлів мережі.

*Telnet* – протокол, що забезпечує передачу потоку байтів між процесами або між процесом і терміналом, який зазвичай використовується для емуляції термінала віддаленої станції.

*SSH (Secure Shell Protocol)* є аналогом протоколу Telnet, але при цьому здійснюється шифрування даних для передачі.

*SMTP (Simple Mail Transfer Protocol)* – простий протокол передачі пошти, який використовується для забезпечення передачі електронних поштових повідомлень із застосуванням транспортного протоколу TCP.

**Стек протоколів IPX/SPX**

Цей стек є оригінальним стеком протоколів фірми Novell, розробленим для мережевої операційної системи NetWare ще на початку 80-х років. Протоколи Internetwork Packet Exchange (IPX) і Sequenced Packet Exchange (SPX), які й дали назву стеку, є прямою адаптацією протоколів XNS фірми Xerox, поширених набагато менше, ніж стек IPX/SPX. Популярність стека IPX/SPX безпосередньо пов'язана з операційною системою (ОС) Novell NetWare.

Цей стек орієнтувався на роботу в локальних мережах невеликих розмірів, які мають невеликі обчислювальні потужності, тому протоколи IPX/SPX мають свої особливості (табл. 3).

Таблиця 3 – Стек протоколу IPX/SPX

| Рівні моделі OSI | Протоколи IPX/SPX |
|---|---|
| 7 | NCP, SAP |
| 6 | |
| 5 | |
| 4 | SPX |
| 3 | IPX, RIP, NLSP |
| 2 | Підтримуються всі популярні стандарти |
| 1 | |

На рівні, який відповідає фізичному й канальному рівням моделі OSI, стек IPX/SPX підтримує всі популярні протоколи цих рівнів.

Наступний рівень, який виконує функції мережевого рівня моделі OSI, реалізований протоколами IPX, RIP і NLSP.

*IPX (Internetwork packet exchange)* – міжмережевий обмін пакетами – протокол, що регламентує обмін даними мережею і працює за дейтаграмним принципом, тобто без встановлення попереднього логічного з'єднання, що забезпечує більш економне споживання обчислювальних ресурсів.

*RIP (Routing Information Protocol)* – протокол маршрутної інформації, являє собою один з найстарших протоколів, які реалізують процеси обміну маршрутною інформацією, однак він дотепер надзвичайно розповсюджений в обчислювальних мережах.

*NLSP (Netware Link Servies Protocol)* – протокол керування зв'язками NetWare – протокол, розроблений під операційні системи NetWare, який забезпечує передачу даних і дозволяє вибирати оптимальні маршрути в мережі.

На рівні, який відповідає транспортному, використовується протокол SPX, що дав частину назви стеку, у якому він і використовується.

*SPX (Sequenced Packet exchange)* – упорядкований обмін пакетами – комунікаційний протокол, розроблений для використання в мережах NetWare. SPX працює з встановленням логічного з'єднання й забезпечує гарантовану доставку й порядок повідомлень у потоці пакетів, для посилки яких використовує протокол IPX.

На верхніх рівнях використовуються протоколи NCP і SAP.

*NCP (Netware Core Protocol)* – основний протокол для передачі інформації між сервером NetWare і робочою станцією. За допомогою функцій цього протоколу робоча станція підключається до серверу, має можливість переглянути файлову систему серверу, копіює віддалені файли, здійснює розподіл мережевого принтера між робочими станціями тощо.

*SAP (Service Advertising Protocol)* – протокол оголошення про сервіс, за принципом дії подібний протоколу RIP. Аналогічно з тим, як різні вузли мережі обмінюються маршрутною інформацією за допомогою протоколу RIP, мережеве обладнання одержує можливість обмінюватися інформацією про наявні мережеві сервіси, використовуючи протокол SAP.

На сьогоднішній день стек IPX/SPX реалізований не тільки в NetWare, але й у декількох інших популярних мережевих ОС, наприклад Microsoft

Windows. Починаючи з версії 5.0 фірма Novell в якості основного протоколу своєї серверної операційної системи стала використовувати протокол TCP/IP, і з того часу практичне застосування IPX/SPX стало неухильно знижуватися.

**Стек протоколів NetBIOS/SMB**

*Стек NetBIOS/SMB* – спільний проект компаній Microsoft та IBM, розроблений у 1984 р. (табл. 4).

Стек працює з усіма найбільш розповсюдженими протоколами нижнього рівня.

На верхніх рівнях працюють протокол NetBEUI та SMB.

Протокол NetBIOS (Network Basic Input/Output System) став розширенням стандартних функцій базової системи введення/виведення (BIOS – Base Input/Output System), який забезпечує підтримку роботи в мережі. У подальшому NetBIOS був замінений протоколом NetBEUI. При цьому NetBIOS все ж був збережений для забезпечення сумісності додатків.

Таблиця 4 – Стек протоколів NetBIOS/SMB

| Рівні моделі OSI | Протоколи NetBIOS/SMB |
|---|---|
| 7 | SMB |
| 6 | |
| 5 | NetBIOS, NetBEUI |
| 4 | |
| 3 | |
| 2 | Підтримуються всі популярні стандарти |
| 1 | |

*NetBEUI (NetBIOS Extended User Interface)* – протокол розширеного користувальницького інтерфейсу NetBIOS, який надає функції, що відносяться до сеансового, транспортного і частково до мережевого рівнів моделі OSI. NetBIOS підтримує як дейтаграмний спосіб обміну даними, так і обмін із установленням логічних з'єднань. Однак цей протокол не забезпечує маршрутизацію пакетів, тому його застосування обмежується тільки невеликими локальними мережами. Для вирішення цієї проблеми використовується NBF (NetBEUI Frame) – реалізація цього протоколу, який вперше з'явився в операційній системи Microsoft Windows NT. Проте в складних мережах використовують більш універсальні протоколи стеків TCP/IP та IPX/SPX.

*SMB (Server Message Block)* – протокол, який виконує функції прикладного рівня і рівня представлення моделі OSI, визначає взаємодію робочої станції та сервера. SMB надає основні мережеві сервіси, необхідні додаткам: керування сесіями передачі даних, встановлення та ліквідацію логічного з'єднання, доступ для роботи з файлами, друк по мережі, передачу повідомлень тощо.

**Розбіжності і особливості поширених протоколів**

Протоколи, що використовуються для обміну даними в локальних мережах, поділяються за своєю функціональністю на три типи:

- прикладні;
- транспортні;
- мережеві.

*Прикладні протоколи* виконують функції трьох верхніх рівнів моделі OSI – прикладного, рівня представлення і сеансового. Вони забезпечують взаємодію додатків і обмін даними між ними. До найбільш популярних прикладних протоколів відносяться:

- *FTAM (File Transfer Access and Management)* – протокол OSI доступу до файлів;
- *X.400* – протокол OSI для міжнародного обміну електронною поштою;
- *X.500* – протокол OSI служб файлів и каталогів на декількох системах;
- *SMTP (Simple Mail Transfer Protocol)* – протокол Інтернету для обміну електронною поштою;
- *FTP (File Transfer Protocol)* – протокол Інтернету для передачі файлів;
- *Telnet* – протокол Інтернету для реєстрації на віддалених серверах і обробки даних на них;
- *SMB (Server Message Blocks)* – протокол взаємодії робочої станції і серверу фірми Microsoft;
- *NCP (NetWare Core Protocol)* – протокол передачі даних між сервером NetWare и робочою станцією фірми Novell;
- *Apple Talk и Apple Share* – набір мережевих протоколів фірми Apple;
- *AFP (AppIeTalk Filling Protocol)* – протокол віддаленого доступу до файлів фірми Apple;
- *DAP (Data Access Protocol)* – протокол доступу до файлів мереж DECnet.

*Транспортні протоколи* реалізують функції транспортного і сеансового рівня моделі OSI. Вони ініціюють і підтримують сеанси зв’язку між вузлами

мережі і забезпечують необхідний користувачам рівень надійності передачі даних. Найпопулярніші серед них наступні:

- *TCP (Transmission Control Protocol)* – протокол Інтернету для гарантованої доставки даних, розбитих на послідовність фрагментів;
- *SPX (Sequential Packet Exchange)* – протокол стеку IPX/SPX для передачі даних, розбитих на послідовність фрагментів, фірми Novell;
- *NetBIOS (Network Basic Input/Output System)* – протокол встановлення і контролю сеансів зв'язку між комп'ютерами;
- *ATP (AppleTalk Transaction Protocol), NBP (Name Binding Protocol)* – протоколи сеансів зв'язку і транспортування даних фірми Apple.

*Мережеві протоколи* виконують функції трьох нижніх рівнів моделі OSI – мережевого, канального й фізичного. Ці протоколи управляють адресацією, маршрутизацією, перевіркою помилок і повторною передачею кадрів, забезпечуючи послуги зв'язку, і визначають правила здійснення зв'язку в окремих середовищах передачі даних, наприклад, Ethernet або Token Ring. До найпопулярніших мережевих протоколів відносяться:

- *IP (Internet Protocol)* – протокол Інтернету для передачі пакетів;
- *IPX (Internetwork Packet Exchange)* – протокол для передачі і маршрутизації пакетів фірми Novell;
- *NetBEUI* – транспортний протокол, що забезпечує послуги транспортування даних для сеансів і додатків NetBIOS фірми Microsoft;
- *DDP (Datagram Delivery Protocol)* – AppleTalk-протокол транспортування даних фірми Apple.

Крім особливостей, обумовлених виконуваними функціями, відмінності і особливості протоколів характеризуються їхньою орієнтацією на роботу в різних операційних системах і з різними апаратними платформами.

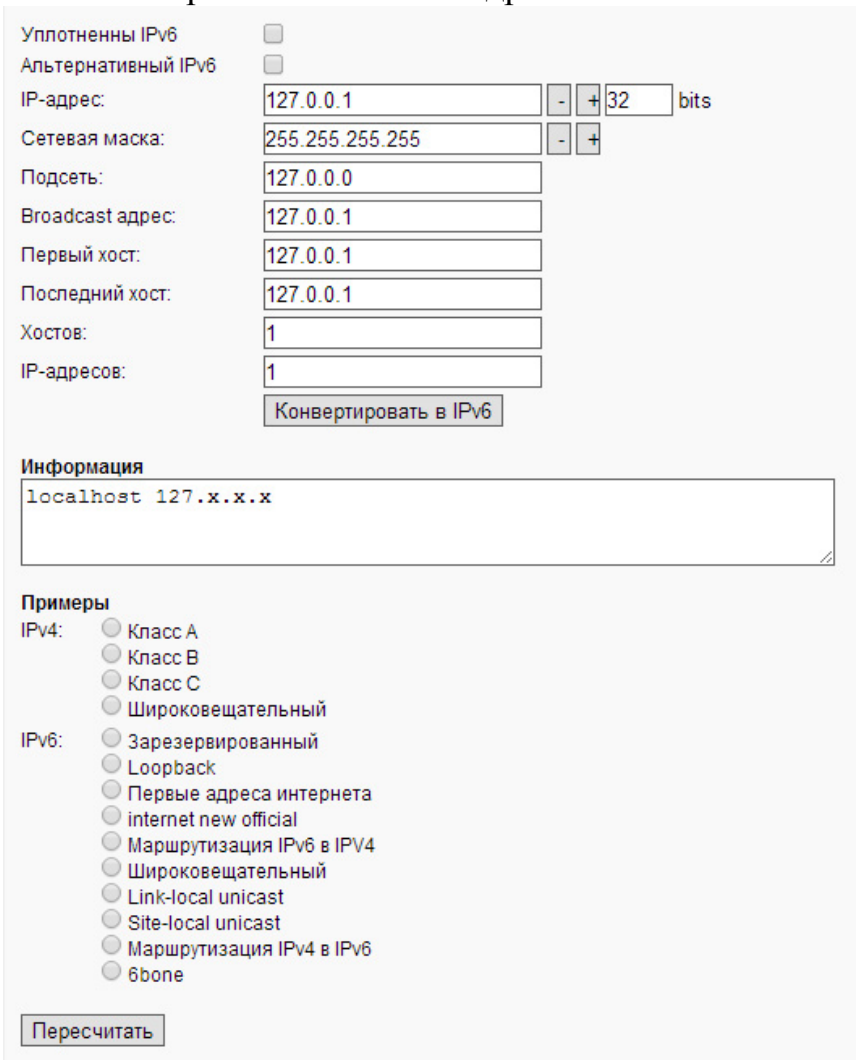
У ході обміну даними мережею протоколи різних рівнів тісно взаємодіють один з одним. Протоколи більш високих рівнів використовують можливості й сервіси протоколів нижніх рівнів.

Додатки обмінюються інформацією за допомогою засобів, що надаються прикладними протоколами, які, у свою чергу забезпечують передачу даних за рахунок використання відповідних транспортних протоколів.

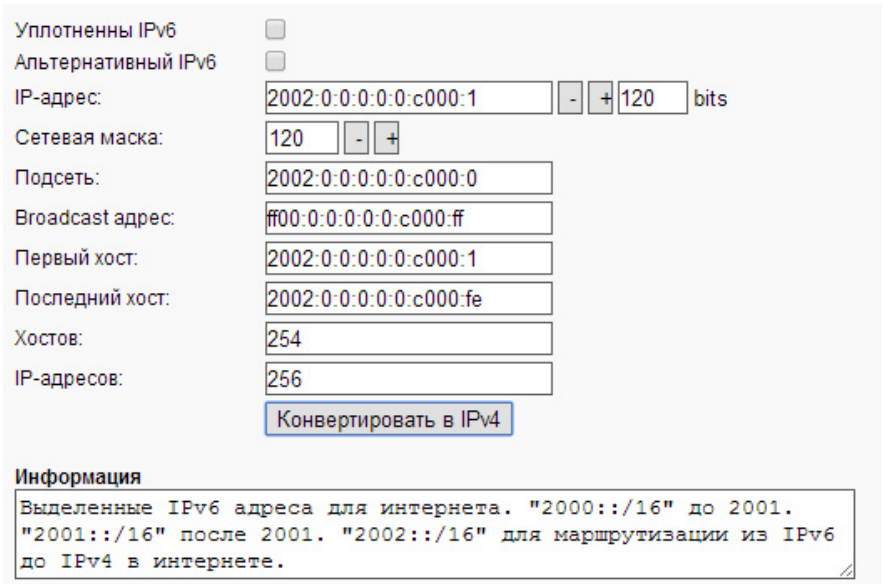
Транспортні протоколи здійснюють передачу даних, використовуючи послуги мережевих протоколів, відповідальних за керування адресацією, маршрутизацію в мережах, забезпечення надійності передачі даних тощо.

**Завдання для Лабораторної роботи №4**

Для перекладу адреси стандарту IPv4 в IPv6 і назад, з можливістю розрахунку мережевий маски, підмережі, першого і останнього хоста діапазону надається додаткова довідкова інформація за класами мереж, що працюють з альтернативними IPv6 адресами. Розглянемо калькулятор.




Приклад конвертації в протокол IPv6:



1) Для заданої IP-адреси перевірити її працездатність за допомогою консолі та команди ping.
2) Для заданої IP-адреси версії IPv4 визначити за допомогою калькулятора альтернативну конфігурацію IP-адреса IPv6 та перевірити працездатність при різних конфігураціях (Зарезервований, широкомовний, перші адреси Internet та інші).
3) Зробити скріни налаштувань, використовуючи задану IP-адресу, згідно свого варіанту.

## 8.Функції мережевого програмного забезпечення

## (Завдання для Лабораторної роботи №5)

Системне програмне забезпечення (ПЗ) повинно підтримувати роботу з мережами відповідної топології та обмін даними з використанням протокольного стеку TCP/IP. Таких операційних систем використовується кілька.

З програмних продуктів фірми Microsoft розповсюджені в Україні системи Windows NT та Windows 2000. Ці системи дозволяють сегментувати мережу, підтримують роботу з Web-серверами, мають потужні засоби архівування і реплікації даних та захисту від несанкціонованого доступу за допомогою ідентифікаційних кодів.

Операційна система NetWare фірми Novell досить розповсюджена, але її використовують у професійних цілях, оскільки, а її адміністрування значно складніше.

Нарешті, у вузлах глобальних мереж найчастіше використовують операційну систему Unix.

### Прикладне мережеве програмне забезпечення

Використання прикладних систем у мережах має певні переваги перед їх автономним використанням. За умови наявності надійного зв'язку з сервером і сучасного апаратного забезпечення на робочих місцях користувачів прикладні системи інсталюють у мережевому варіанті. При цьому ядро програми інсталюється на сервері, а на робочих місцях інсталюється лише клієнтська частина. Такий варіант інсталяції передбачений і в пакеті MS Office починаючи з версії 2000. Зауважимо, що при використанні мережевих версій прикладних програм дуже бажано використовувати окремий виділений файл-сервер для зберігання файлів, з якими працюють кілька користувачів одночасно.

При такому використанні прикладних програм не має значення, де саме інсталюється серверна частина програми - на сервері локальної мережі, чи на сервері провайдера Internet. В останньому випадку йдеться про аутсоринг програмного забезпечення - оренду.

Такий спосіб використання програмного забезпечення має наступні переваги:

- зменшення витрат на придбання програмного забезпечення. Один примірник ліцензії на використання мережевого варіанту програмного забезпечення коштує дешевше, ніж кілька примірників ліцензії на використання автономних версій програми;
- надання можливості кільком користувачам одночасно працювати з одним документом. При цьому всі зміни фіксуються в одному файлі. Фіксується також участь різних користувачів у створенні документу як за змістом, так і за часом;
- у випадку періодичної оренди програмного забезпечення ліцензію на його використання взагалі не потрібно купувати. В цьому випадку оплачується лише вартість послуг.

Зрозуміло, що мережеве використання прикладних систем має і суттєві недоліки. У випадку порушень роботи мережі (порушення в роботі сервера, порушення зв'язку) робота користувачів блокується повністю.

Найвживанішими окремими програмами-клієнтами мереж є браузери і поштові клієнти. Хоча в пакеті MS Office передбачені клієнти для доступу до ресурсів глобальних мереж і для надання поштових послуг (Outlook Express), доцільніше використовувати програми спеціального призначенн. Як браузер використовують або NetscapeNavigator, або Mozilla. Як поштовий клієнт часто використовують програму TheBat, яка є вільним продуктом.

Окремо, зазвичай, використовують і спеціальні безкоштовні програми для отримання файлів з серверів мережі, хоча такі можливості має кожна програма-браузер. Це зайвий раз доводить визнану перевагу використання спеціального програмного забезпечення перед універсальним. Таких програм є дуже багато, тому перераховувати їх тут немає потреби.

У світі набуло розповсюдження спеціальне мережеве програмне забезпечення (з використанням технології клієнт-сервер) для надання торгівельних, освітніх, бібліотечних, комерційних, баківських послуг. Ці системи та відповідне програмне забезпечення і технології отримали назву електронних. Такі технології дозволяють, маючи доступ до Internet в режимі on-line, формувати і передавати на відповідний сервер (сервер організації, що надає потрібні послуги) запит на надання послуги, пошук необхідних даних

чи виконання відповідних операцій (банківських, купівлі-продажу) та отримати повідомлення про виконання запиту. Зазвичай, використання таких систем і технологій передбачає використання системи електронних розрахунків, за відсутності якої їх ефективність різко зменшується.

**Завдання для Лабораторної роботи №5**

1) За допомогою служби мережевих команд перевірити працездатність мережевого ПЗ: Outlook, The Bat.
2) За допомогою мережевого обладнання створити обчислювальну мережу.
3) Пояснити принцип маркування мережевих кабелів.
4) Зробити скріни роботи з ПЗ та обладнанням.

## Література


1. Фельдман Л.П., Петренко А.І., Дмитрієва О.А. Чисельні методи в інформатиці, Київ, видавнича група BHV,2006.
2. Бахвалов Н.С., Жидков Н.П., Кобельков Г.М. Численные методы, Москва, Наука, 1987.
3. Самарський А.А., Гулин А.В. Численные методы, Москва, Наука, 1989, 430с.
4. Шуп Г.Е. Прикладные численные методы в физике и технике, Москва, Высшая школа, 1990.
5. Березин И.С., Жидков Н.П. Методы вичислений, ФИЗМАТГИЗ, Москва, 1960.
6. Воробьева Г.Н., Данилова А.Н. Практикум по вычислительной математике, Москва, Высшая школа, 1990.

**ЗМІСТ**

Підписано до друку 15.03.2016. Формат 60x84/16.
Гарнітура TimesNewRoman. Ум. друк. арк. 2,5.
Наклад 100 прим. Віддруковано на різографі.

---

*Видавництво УжНУ «Говерла»*
*88000, м. Ужгород, вул. Капітульна, 18.*
*Свідоцтво про внесення до державного реєстру видавців виготівників, і розповсюджувачів видавничої продукції*

*Серія Зт №32 від 31 травня 2006 року*