Курс «Паралельні алгоритми: побудова та аналіз».

Лекція 4. ***Підходи до побудови паралельних алгоритмів.***

План лекції.

1. Автоматичне та автоматизоване розпаралелювання.
2. Поняття статичного та динамічного розпаралелювання.
3. Основні етапи розпаралелювання ациклічних ділянок.
4. Розпаралелювання циклів: розпаралелюючий компілятор.

1. Розроблення паралельної програми є зазвичай складною, висококваліфікованою і непродуктивною працею. Паралельні програми з великими затратами переносяться на різні архітектури паралельних обчислювальних систем. Коло користувачів багатьох паралельних програм є доволі вузьким. Такі обставини були і залишаються потужним стимулом для створення **засобів автоматичного розпаралелювання** послідовних програм та алгоритмів.

Програмування на **послідовній мові** і подальше автоматичне розпаралелювання виглядає досить привабливо, оскільки дозволяє:

- застосовувати досвід програмування, набутий протягом багатолітньої експлуатації традиційних послідовних ЕОМ;
- розробляти програми, що не залежать від особливостей архітектури паралельної обчислювальної системи;
- використовувати програмні продукти, накопичені за багато років створення програмного забезпечення для послідовних ЕОМ;
- не витрачати сил та засобів на паралельну організацію програм, вважаючи, що система автоматичного розпаралелювання вирішить усі питання з ефективного перенесення програм на будь-яку паралельну систему.

Однак, зараз послідовне програмування і подальше автоматичне розпаралелювання **не часто** використовуються в **промисловому програмуванні**, оскільки не завжди забезпечують досягнення прийнятного рівня ефективності паралельних програм.

Можна виділити чотири **етапи розв'язання деякої задачі** на паралельній обчислювальній системі:

- вибір паралельного алгоритму, що найбільше підходить для розв'язання цієї задачі (**етап А**);
- подання алгоритму на мові високого рівня (яка найбільше підходить для даного випадку) – написання паралельної програми (**етап М**);
  мови високого рівня (Fortran, C, C++, C#, Java, Python, Eiffel, Ruby тощо) виражають потреби програміста, а не можливості комп'ютера;
  мова низького рівня (асемблер, машинний код) є близькою до програмування безпосередньо в машинних кодах використовуваного процесора;
- компіляція програми в об'єктний код (**етап О**);
- виконання паралельної програми (**етап П**).

Ступені паралелізму результатів наведених етапів (алгоритму, програми, об'єктного коду, обчислювального процесу) позначимо відповідно $\|A, \|M, \|O, \|П$. Очевидно, що справджуються наступні **нерівності**:

$$\|A \geq \|M \geq \|O \geq \|П.$$

Проблема полягає в тому, щоб не розгубити, а максимально **зберегти паралелізм** алгоритму А на шляху:

$$A \to M \to O \to П.$$

Паралелізм А є **максимально досяжним** (потенційним) паралелізмом для всіх етапів. Наступні рисунки (див. рис. 1 – 4) ілюструють різні варіанти розв'язання задач на паралельних обчислювальних системах з погляду збереження паралелізму алгоритму А на шляху $A \to M \to O \to П$.

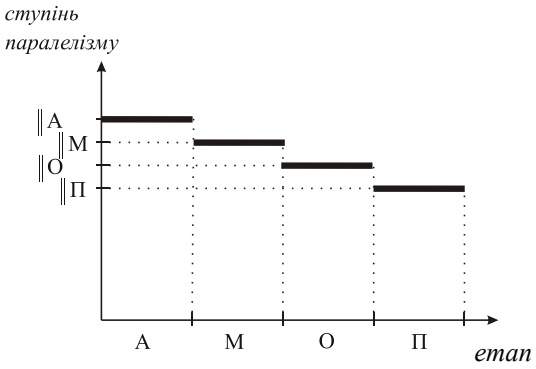

Рис. 1. Ідеальний варіант збереження паралелізму

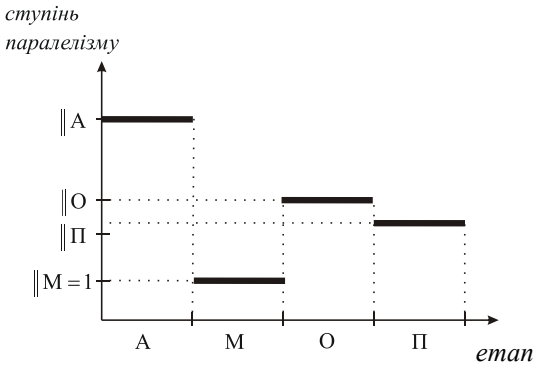

Рис. 2. Втрати та відновлення паралелізму під час використання послідовної мови програмування і розпаралелюючого, зокрема, векторизуючого компілятора

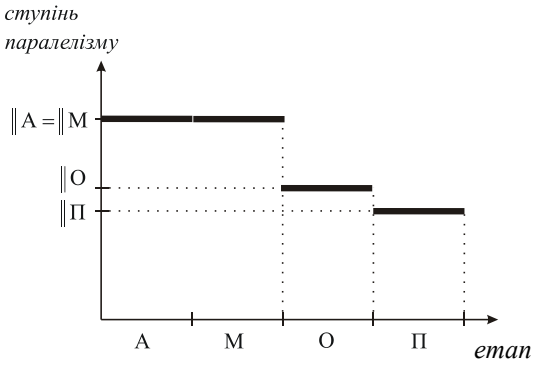

Рис. 3. Використання проблемно-орієнтованої (непроцедурної) мови програмування

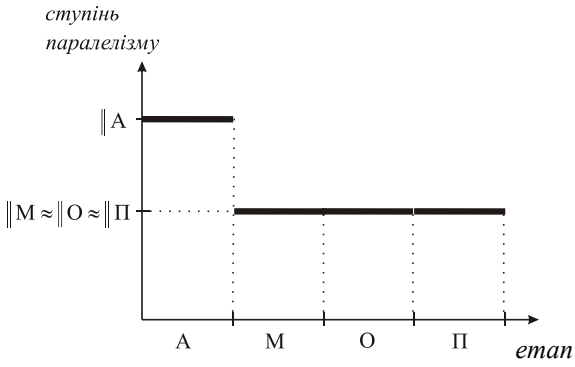

Рис. 4. Використання паралельної мови високого рівня, орієнтованої на архітектуру заданої паралельної обчислювальної системи

Непроцедурна мова (SQL, Lisp, ML, Prolog тощо) дозволяє описувати мету і правила, а не послідовність дій з їх реалізації, тобто описується «що робити» замість «як робити».

Поряд з **автоматичним розпаралелюванням** в обчислювальній практиці широко використовується **автоматизоване розпаралелювання** послідовних програм або алгоритмів, коли програміст «підказує» компілятору, які фрагменти слід розпаралелити і яким методом (у разі розпаралелювання, наприклад, циклів).

2. Розрізняють **статичне та динамічне розпаралелювання** послідовних програм. Найчастіше використовують **статичне** розпаралелювання (розпаралелювання до виконання програми), яким ми і обмежимося. Під час **динамічного** розпаралелювання послідовних програм програма аналізується **в процесі виконання**. На підставі цього аналізу приймається рішення про призначення різних операторів для паралельного оброблення на відповідних пристроях.

Взагалі кажучи, розпаралелювання програм або алгоритмів можна здійснювати на **різних рівнях** (задач, процедур, операторів (виразів), операцій, мікроопераций (бітовий рівень)). Доцільність того чи іншого рівня розпаралелювання вирішується для кожного окремого випадку залежно від структури обчислювальної системи, мови програмування та мети розпаралелювання.

Є велика кількість **комерційних** і **некомерційних** систем **автоматичного і автоматизованого розпаралелювання** послідовних програм. Для прикладу розглянемо пакет PGI, запропонований компанією Portland Group. Цей пакет містить набір оптимізуючих і розпаралелюючих компіляторів та засобів для полегшення процесу розробки і підтримки обчислювальних застосувань на робочих станціях, серверах і кластерах на базі процесорів Intel:

- PGHPF – компілятор з мови HPF (High Performance Fortran – високопродуктивний фортран, підтримує стиль програмування, який використовує паралелізм оброблення даних);
- PGF90 – компілятор Фортрану 90;
- PGF77 – компілятор Фортрану 77;
- PGCC – компілятори ANSI C і C++ (ANSI – Американський Національний Інститут Стандартизації; майже весь код, написаний останнім часом, відповідає цьому стандарту).

Цей пакет працює в операційних системах Linux, Solaris і Windows.

Серед **закордонних** оптимізуючих і розпаралелюючих **компіляторів** можна виділити:

GNU Compiler Collection (GCC)
(підтримує C, C++, Java, Ada, Go, Fortran; є вільним програмним забезпеченням),
Microsoft Visual C++
(ліцензійне забезпечення, однак є вільні версії),
ROSE
(підтримує C, C++, Fortran, Java, Python; здійснює автоматизоване розпаралелювання, відкритий ресурс),
Oracle Solaris Studio (OSS)
(ліцензійне забезпечення; підтримує Fortran, C, C++, Java),
Intel C++ Compiler

(є вільна версія під Linux; здійснює автоматичне розпаралелювання; підтримує C, C++).

Розробляються **системи розпаралелювання** (фактично це окремі великі проекти з побудови компіляторів):

SUIF Compiler (автоматизоване розпаралелювання; підтримує C),
PLUTO (автоматичне розпаралелювання; підтримує Java),
Par4All (автоматизоване розпаралелювання; підтримує C, Fortran).

Прикладами **російських розробок** в галузі оптимізації і **розпаралелювання** програм є

система **ПРОГРЕСС** (автоматизоване розпаралелювання; підтримує C, Fortran);
DVM-система (автоматизоване розпаралелювання), що означає
Distributed Virtual Memory (наявність єдиного адресного простору) або
Distributed Virtual Machine (відображає використання віртуальних машин);
система **ОРС** (Открытая Распараллеливающая Система);
система **ДВОР** (Диалоговый Высокоуровневый Оптимизирующий Распараллеливатель).

3. Процес розпаралелювання ациклічних ділянок програм або алгоритмів складається із таких основних етапів:

- побудова **графу залежностей** за даними між операторами програми або алгоритму, тобто **побудова графу алгоритму**;
- **побудова ярусно-паралельної форми** програми або алгоритму;
- **написання** на підставі ярусно-паралельної форми **паралельної програми**;
- **відображення** згаданої **програми на архітектуру** наявної паралельної обчислювальної **системи**.

Граф залежностей за даними G між операторами програми (або алгоритму) будується на підставі ***інформаційної, конкуренційної та логічної залежностей.***

Інформаційна залежність між операторами програми або алгоритму. Оператор B залежить від оператора A інформаційно, якщо оператор A «виробляє» значення деякої змінної x, яке використовує оператор B. Іншими словами, оператор B залежить від оператора A інформаційно, якщо:

1) існує шлях від вхідного оператора, що проходить через оператори A, B;
2) оператор A є останнім перед B оператором цього шляху, який «виробляє» значення змінної x, що використовується оператором B.

Формально інформаційну залежність оператора B від оператора A можна записати у такому вигляді:

$$In(B) \bigcap Out(A) \neq \varnothing,$$

де $In(B)$ – сукупність вхідних змінних оператора B; $Out(A)$ – сукупність вихідних змінних оператора A.

Конкуренційна залежність. Оператори A і B залежать один від одного конкуренційно, якщо:

1) існують шляхи від вхідного оператора програми як до оператора A, так і до оператора B;

2) оператори A і B «виробляють» значення однієї і тієї ж змінної x.

Формально конкуренційну залежність операторів A і B можна записати у такому вигляді:

$$Out(A) \bigcap Out(B) \neq \varnothing .$$

Логічна залежність між операторами програми або алгоритму. Оператор B залежить від оператора A логічно, якщо:

1) існує шлях від вхідного оператора програми до оператора A;
2) оператор A є «розпізнавачем», який вирішує: буде чи не буде виконуватись оператор B.

Детальніше побудову **графу алгоритму** ми розглядали на практичних заняттях, до того ж розглядали загалом **детерміновані графи** алгоритму, тобто такі, що не містять умовних операторів.

Ярусно-паралельна форма для програми або алгоритму дозволяє визначити операції, оператори або фрагменти обчислень, які можна виконати одночасно на одному ярусі (кроці).

Побудова паралельної програми за ярусно-паралельною формою. Фактично це є розбиття послідовного алгоритму або програми на підставі їх ярусно-паралельної форми на окремі паралельні процеси (або гілки) з метою виконання на паралельній обчислювальній системі.

Для прикладу, якщо ми маємо паралельну систему зі спільною пам’яттю, то накладні витрати на комунікацію будуть мінімальними, оскільки всі процеси розділяють один адресний простір, а синхронізаційні операції виконуються простіше і швидше, ніж для звичайних (асинхронних) процесів.

Очевидно, що **розбиття** програми або алгоритму на окремі гілки (процеси) є **багатоваріантним**. За критерій оптимального розбиття природно використовувати **час виконання** програми або алгоритму.

Оскільки метою розбиття програми або алгоритму на окремі гілки (процеси) є зменшення часу обчислень, то кількість цих гілок, зазвичай, вибирається такою, щоб дорівнювала кількості процесорів у паралельній обчислювальній системі. З цієї ж причини під час такого розбиття необхідно намагатися, щоб у гілках виконувалась приблизно одинакова кількість операцій (операторів), оскільки існує **проблема збалансованого завантаження процесорів** обчислювальної системи корисною (обчислювальною) роботою. При цьому особливу увагу слід звернути і на **проблему синхронізації гілок**, якщо вони обмінюються даними, бо на це витрачається додатковий час.

Ефективність паралельної програми здебільшого залежить від ефективності використання нею **кеш-пам’яті**, оскільки часто запитувані дані переважно розташовуються у більш швидкій кеш-пам’яті.

Якщо у паралельній програмі кількість гілок (процесів) дорівнює кількості процесорів в обчислювальній системі, на якій ми хочемо цю програму виконати, то етап ***відображення паралельної програми на архітектуру паралельної обчислювальної системи*** не є якоюсь складністю. Якщо ж немає такої відповідності, то паралельну програму потрібно переробляти так, щоб кількість паралельних гілок у ній відповідала кількості процесорів обчислювальної системи.

4. Загалом **розпаралелювання циклів** проводиться **компіляторами**. Якщо мова йде про розпаралелювання для векторноконвеєрної або векторнопаралельної обчислювальних систем, то такий компілятор називається **векторизуючим компілятором** (векторизатором). Аналогічно, якщо йдеться про розпаралелювання циклів для багатопроцесорної обчислювальної системи, то говорять про **розпаралелюючий компілятор**, загальна структура якого наведена на рис. 5.

На етапі **синтаксичного аналізу** проводиться синтаксичний аналіз послідовної програми (алгоритму), перевірка деяких обмежень на тіла циклів (лінійність індексних виразів, відсутність операторів вводу-виводу та звернень до підпрограм тощо). Окрім цього на даному етапі оцінюється час виконання програми. Якщо для такої оцінки недостатньо інформації, то її компілятор може запросити в користувача у діалоговому режимі.

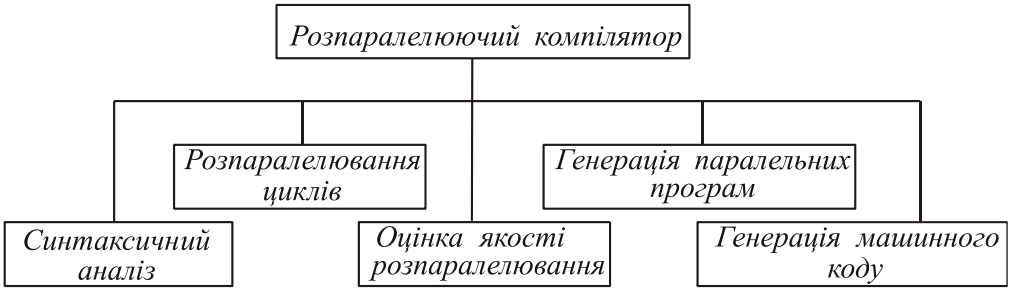

Рис. 5

На етапі **розпаралелювання циклів** здійснюється перевірка виконання решти необхідних обмежень на тіла циклів (зокрема, умови Рассела). Тут же для кожного циклу із багатьох методів, реалізованих в компіляторі, здійснюється вибір методу розпаралелювання. Для циклів, які не розпаралелюються, компілятор видає інформацію про конкретні причини цього.

На етапі **оцінки якості розпаралелювання** оцінюється ступінь розпаралелювання (кількість ітерацій, які можна виконати одночасно) кожного із циклів і на підставі цього здійснюється оцінка прискорення паралельної програми порівняно з вихідною послідовною програмою. Якщо це прискорення є незадовільним, то на підставі інформації, виданої компілятором на попередньому етапі, користувач може спробувати перетворити цикли так, щоб вони могли бути ефективно розпаралелені.

На етапі **генерації паралельної програми** генерується програма на паралельній мові високого рівня.

А на етапі **генерації машинного коду** компілятор генерує код для використовуваної паралельної обчислювальної системи.