УДК 519.685.3

К.т.н., доцент Россошинський Д. О., магістрант Бойко О. А.

Національний технічний університету України
«Київський політехнічний інститут»

АВТОМАТИЧНЕ ПЕРЕТВОРЕННЯ ПОСЛІДОВНИХ ПРОГРАМ ДЛЯ ВИКОНАННЯ НА ОБЧИСЛЮВАЛЬНИХ СИСТЕМАХ З ПАРАЛЕЛЬНОЮ АРХІТЕКТУРОЮ

Abstract

Dmitro O. Rossoshinskiy, assoc. prof., PhD; Olga Boiko, student
Automatic conversion of sequential programs for execution in computer systems with parallel architecture

This paper concerns the task of compilation system, that can find and implement the hidden parallelism of sequential programs and algorithms.

Вступ

Зростання обчислювальних потужностей комп'ютерної техніки підвищує інтерес до неї все більшої кількості галузей людської діяльності. Це в свою чергу спричиняє виникнення нових задач, що потребують ще більших потужностей.

Для вирішення цієї задачі комп'ютерна інженерія розвивається шляхом збільшення кількості ядер в процесорах та об'єднання процесорів. Набуло розвитку паралельне програмування – створення спеціального програмного забезпечення. Створюються спеціальні мови, нові алгоритми, виникає задача переносу вже існуючого програмного матеріалу на нову архітектуру. Для цього використовують спеціалізовані компілятори.

Основна задача розпаралелюючого компілятора – виділити максимум прихованого паралелізму з повторювальних ділянок послідовної програми, що суттєво впливають на час її виконання: циклів та рекурсивних процедур. Це вимагає зміни порядку виконання операторів на цих ділянках, що можливо лише за відсутності залежності по даних між операторами.

Таким чином спершу необхідно виконати аналіз залежностей: визначити, які частини послідовної програми можуть виконуватись паралельно, а які потребують послідовного виконання або іншої форми синхронізації. Залежності по даних легко визначити в послідовному коді, що містить посилання лише на скалярні змінні. Набагато складніше це

зробити в циклах і у випадку посилань на масиви, які, зазвичай, використовуються разом і використовують спільні змінні.

Перевірка залежності в загальному вигляді є складною (а в гіршому випадку – нерозв'язною) задачею. Ефективні перевірки існують лише для часткових випадків. Тому перед розробниками постає задача визначення алгоритму, який має бути застосований [1].

Наступним кроком є реструктуризація програми. Алгоритми розпаралелювання циклів в основному базуються на:

- декомпозиції графа залежностей на сильно-зв'язні компоненти;
- унімодулярних перетвореннях циклу (спеціальних або автоматично згенерованих);
- плануванні (одномірному або багатомірному).

Кожна з цих груп алгоритмів використовує свої математичні інструменти, потребує на вході особливого представлення залежностей по даних, і тому покриває конкретний клас задач. Таким чином, важливою залишається задача пошуку оптимального алгоритму для кожної конкретної задачі [2].

Постановка задачі

Ідея автоматичного розпаралелювання програм ефективно вирішує проблему переносу вже існуючих послідовних програм на паралельну архітектуру. Однак, через складність виявлення і реалізації прихованого паралелізму, досі не існує універсального рішення.

Метою даної роботи є побудова системи-компілятора, що об'єднає різні класи методів пошуку та реалізації прихованого паралелізму в програмах і дозволить аналізувати ефективність виконання алгоритму в паралельній обчислювальній системі.

Для конкретизації задачі розглядаються лише системи з розподіленою пам'яттю. Мова програмування вхідної програми – Сі.

Також необхідно визначити метод перевірки ефективності виконання програм до та після перетворення на різних обчислювальних системах.

Розробка системи розпаралелювання програм

Система має бути модульною. Таким чином, можна буде проводити її подальше удосконалення і масштабування.

Система дозволить аналізувати вхідну програму, проводити її перетворення і отримати на виході паралельну програму.

На рис.1. показана взаємодія компонент розробленої нами системи розпаралелювання програм.

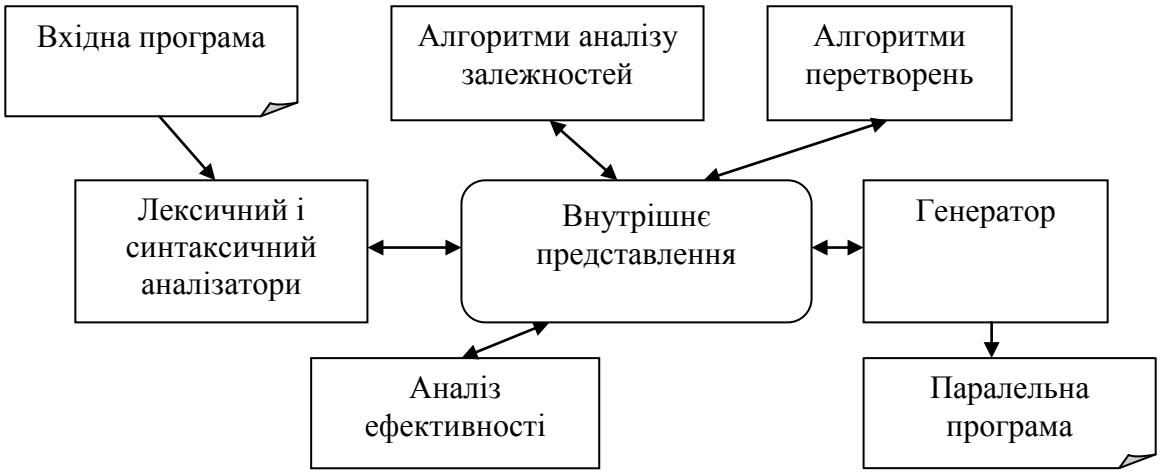

Рис.1. Взаємодія компонент системи

На першому етапі програма проходить лексичний і синтаксичний аналіз і перетворюється в проміжне (внутрішнє) представлення. Воно необхідне для зручності виконання подальших етапів з одного боку і уніфікації коду з іншого.

Внутрішнє представлення, крім самої програми, описує її атрибути, отримані на етапах аналізу залежностей, перетворень та ін. Таким чином, виключається прямий зв'язок між модулями і спрощується робота системи. Було обрано графічне представлення (граф управління програмою) [3].

Блок аналізу залежностей переглядає проміжний код програми для виявлення залежностей по даних і ділянок, де можна її позбутись. Використовуються система тестів, яка складається з різноманітних модифікацій і комбінацій алгоритмів. Починаючи з простих, таких як НСД-тест, на початку, і закінчуючи більш інтелектуальними, заснованими на теорії лінійного програмування, такими як Омега-тест [1]. На виході отримується інформація, що описує прихований паралелізм програми. Ці дані також додаються у внутрішнє представлення і використовуються на етапі перетворень.

Алгоритми перетворень на основі внутрішнього представлення і даних, отриманих в блоці аналізу, дозволяють отримати паралельний варіант програми. В цьому блоці виконуються перетворення програми для паралельного виконання операторів:

- послідовного коду – найпростіший випадок, коли найлегше виявити залежність, паралельність досягається локалізацією змінних, які використовуються кількома процесами;

- умовних операторів – визначення паралелізму і перетворення виконується за допомогою використання теорії графів (приведення графу до плаского вигляду);
- циклів (в загальному випадку вкладених) – найскладніший і найважливіший випадок, оскільки більшість обчислень виконується саме в циклах. Використовуються різні комбінації алгоритмів Аллена і Кеннеді, Вольфа і Лема, Дарте і Вів'єн [2].

Блок визначає потенційні ділянки коду, що можуть бути розпаралелені та пропонує варіанти розв'язку задачі. Вони, в свою чергу, можуть бути виконані за запитом користувача.

До інших алгоритмів, що використовуються в системі, можна віднести алгоритми аналізу продуктивності, що дозволять проводити аналіз роботи і порівняння використаних методів. Цей блок аналізує завантаження процесорів і використання пам'яті під час виконання програми, що дозволяє підібрати найоптимальніший підхід до роботи.

Для аналізу ефективності пропонується використовувати такі показники як ширина та висота алгоритму [3].

Фінальним етапом розпаралелювання є етап генерації. На основі проміжного коду і вибраної користувачем цільової платформи будується готова паралельна програма – файл для виконання.

Висновки

У даній роботі запропоновано концепцію побудови системи-компілятора для автоматичного перетворення програм на послідовному коді для виконання на паралельних обчислювальних системах з розподіленою пам'яттю.

Модульна архітектура дозволяє в подальшому додавати нові бібліотеки для підтримки інших мов програмування та алгоритмів аналізу і перетворення структури програм.

Література

1. *Евстигнеев В. А., Арапбаев Р. Н., Осмонов Р. А.* Анализ зависимостей: основные тесты на зависимость по данням.// Сибирский журнал вычислительной математики, 2007, т. 10, №3, с. 247-265.
2. Программные средства и математические основы информатики. – Новосибирск: Институт систем информатики им. А.П. Ершова СО РАН, 2004. – 278 с.
3. *Воеводин В. В., Воеводин Вл. В.* Параллельные вычисления. – СПб.: БХВ-Петербург, 2002. – 608 с.: ил.