УДК 004.424

РОЗПАРАЛЕЛЮВАННЯ ПРОГРАМ НА ФОРТРАНІ З ВИКОРИСТАННЯМ ТЕХНІКИ ПЕРЕПИСУВАЛЬНИХ ПРАВИЛ

А.Ю. Дорошенко, К.А. Жереб, Є.М. Туліка

Інститут програмних систем НАН України,
проспект Академіка Глушкова, 40,
тел.: 526 1538, e-mail: doroshenkoanatoliy2@gmail.com, zhereb@gmail.com, vranen@gmail.com

Описано підхід до розпаралелювання програм на Фортрані, заснований на використанні техніки переписувальних правил. Для синтаксичного аналізу програмного коду на Фортрані використано засоби компілятора GCC, що дозволяє працювати з кодом на старих версіях Фортрану. Підхід застосовано для розпаралелювання прикладної задачі з галузі квантової хімії. Результати обчислювальних експериментів продемонстрували ефективність запропонованого підходу.

We describe an approach to parallelize Fortran programs based on rewriting rules technique. We use GCC compiler to parse Fortran source code which allows processing code in older versions of Fortran. We have applied our approach to parallelize an applied problem from quantum chemistry domain. The results of computational experiments demonstrate efficiency of proposed approach.

Вступ

Фортран є однією з перших мов програмування, проте ця мова дотепер широко використовується, особливо для вирішення фізичних, хімічних і інших наукових задач [1]. Популярність Фортрану пояснюється його відносною простотою, близькістю вихідного коду до математичного формулювання задачі, ефективністю створюваного бінарного коду. Крім того, за більш ніж 50 років існування мови накопичено величезну кількість програмного забезпечення у вигляді готових програм і бібліотек для вирішення різноманітних задач.

Більша частина існуючого коду була розроблена для послідовних обчислювальних систем. Сучасний розвиток високопродуктивних обчислювальних систем дає потенційну можливість суттєво підвищити продуктивність таких програм за рахунок використання паралельних обчислень [1–2]. Однак повне переписування існуючих програм для нових апаратних архітектур практично неможливо через їх великий обсяг. Тому актуальною є задача розпаралелювання існуючих програм на Фортрані, при цьому особливо важливо якнайбільше автоматизувати процес розпаралелювання.

У даній роботі описано підхід до розпаралелювання програм на Фортрані, заснований на використанні техніки переписувальних правил. Раніше розроблений авторами інструментарій переписувальних правил [3–6] був розширений з метою підтримки цільової мови Фортран, зокрема досить старих версій мови, таких як Фортран 77. Описано використання компілятора Фортран GCC [7] для проведення синтаксичного аналізу тексту програми, а також системи переписувальних правил Termware [8, 9] для автоматизації перетворення послідовної програми в паралельну. Описаний підхід застосовано до прикладної задачі з галузі квантової хімії – програми обчислення електронної та спінової густини [10, 11]. Використання запропонованого підходу дозволило досягти значного підвищення продуктивності паралельної програми за часом виконання.

Робота є продовженням досліджень авторів, що описані в роботах [3–6]. Зокрема, використовується підхід до розпаралелювання, заснований на використанні техніки переписувальних правил та алгебраїчних моделей програм, що аналогічний до описаного в роботі [3]. Також використовується інструментарій переписувальних правил, описаний в роботі [4]. Основна відмінність даної роботи від попередніх робіт [3–5] полягає в орієнтації на іншу мову програмування – Фортран. Автори вже розглядали цю мову в роботі [6]. Відмінність даної роботи від роботи [6] полягає в двох основних напрямках. По-перше, використовується інший засіб синтаксичного аналізу – компілятор GCC – що дозволяє працювати з програмами, написаними на старих версіях мови Фортран. По-друге, запропонований підхід застосовано до реальної прикладної задачі з області квантової хімії [10], на відміну від досить простого прикладу алгоритму Гаусса, який розглядався в [6].

Матеріал даної роботи організовано наступним чином. У розділах 1–2 описано сучасні засоби аналізу коду на мові Фортран, зокрема обраний для даної роботи компілятор GCC. Розділ 3 описує загальний підхід, що застосовується для розпаралелювання програм. Більш детальний опис окремих етапів розпаралелювання наведено в розділах 5–7. Розділ 8 описує прикладну програму, на прикладі якої продемонстровано роботу підходу, а також містить результати порівняння продуктивності послідовної та розпаралеленої програми, що демонструє достатню ефективність запропонованого методу. Роботу завершують висновки та напрямки подальших досліджень.

1. Засоби аналізу коду на Фортрані

Як вже згадувалось, історія мови Фортран налічує багато десятків років, і ця мова користується широкою популярністю в наукових та інженерних галузях. Проте існує не так багато інструментальних засобів, які могли б використовуватись для аналізу коду на Фортрані (зокрема, для розпаралелювання). Це пояснюється складністю самої мови для синтаксичного аналізу, оскільки багато мовних конструкцій орієнтовані на застарілі

©А.Ю. Дорошенко, К.А. Жереб, Е.М. Туліка, 2012

засоби вводу інформації (орієнтація на рядки, спеціальне значення символів в певних позиціях, можливість пропускати пробіли та інші). Крім цього, існує багато версій мови (з використовуваних на даний момент – Фортран 77, 90, 95, 2003, 2008), нестандартних розширень мови, характерних для певних компіляторів, а також різних стилів написання програм (наприклад, fixed form – коли перші 6 символів кожного рядка мають спеціальне значення і не використовуються для коду, та free form – більш нова форма запису без подібних обмежень). Тому вибір засобів аналізу коду на Фортрані є досить складною задачею.

Найбільш потужними інструментальними засобами, які підтримують більшість можливостей мови – як сучасних, так і застарілих – є компілятори. На даний момент існує досить багато компіляторів мови Фортран, як комерційних, так і відкритих. Найбільш популярними з комерційних продуктів є Intel Fortran Composer [12] та PGI Fortran Compiler [13]. Обидва продукти підтримують більшість можливостей різних версій мови Фортран, а також програми, що містять фрагменти різних версій мови. (Досвід авторів показує, що Intel Fortran Composer потребує мінімальних змін при перенесенні старих програм, тоді як PGI Fortran Compiler інколи не підтримує деякі застарілі можливості). Значною перевагою цих продуктів є можливість автоматичного розпаралелювання послідовних програм для мультиядерних паралельних платформ зі спільною пам'яттю. PGI Fortran Compiler також дозволяє розпаралелювати програми для відеографічної платформи, проте ці можливості в даній роботі не розглядались. Але суттєвим недоліком комерційних продуктів є їх закритість, зокрема неможливість отримання результатів синтаксичного аналізу для подальшого використання в інших інструментальних засобах.

З відкритих компіляторів мови Фортран найбільш популярним є компілятор, що входить до набору компіляторів GCC [7]. GNU Compiler Collection (GCC) – система компіляції різноманітних мов (зокрема, C, C++, Objective-C, Fortran, Ada, Go), яка виступає базовим інструментальним засобом для компіляції програм на Unix-подібних системах. Компілятор Фортрану GCC підтримує більшість можливостей мови Фортран версій 77–95 і багато можливостей більш нових версій 2003 та 2008, хоча на відміну від компіляторів Intel Fortran Composer та PGI Fortran Compiler не підтримуються деякі нестандартні розширення, характерні для діалектів Фортрану для операційних систем DOS та Windows. Великою перевагою компіляторів GCC є їх відкритість, модульна архітектура, яка дозволяє легко розширяти систему, та можливість отримання проміжних представлень програми, зокрема дерева синтаксичного розбору. Саме тому в даній роботі було використано компоненти компілятора GCC для синтаксичного аналізу коду програм на Фортрані. На жаль, компілятори GCC не підтримують автоматичне розпаралелювання програм на Фортрані.

Крім компіляторів, існують і інші засоби, орієнтовані на аналіз коду на Фортрані. Зокрема, певною популярністю користуються засоби для трансляції Фортрану в більш розповсюджені на даний момент мови: F2C (Fortran to C) та F2J (Fortran to Java) [14]. Основним мотивом створення транслятора F2J було надання програмного забезпечення для чисельної лінійної алгебри, написаного на Фортрані, у вигляді класів мови Java. Бібліотеки для обчислень розповсюджуються у вигляді файлів класів, створених за допомогою транслятора F2J. F2J формально є компілятором, що транслює програму, написану на мові Фортран 77, в форму, яка може виконуватись на віртуальній машині Java. Першопріоритетною задачею для F2J був переклад чисельних бібліотек BLAS та LAPACK [1]. Також F2J використовувався для створення коду на Java для бібліотеки ARPACK [1]. Засоби F2J орієнтовані на роботу з цими трьома бібліотеками, тому можливості використання для інших проектів суттєво обмежені.

F2C – інструментальний засіб для конвертації програм на мові Фортран 77 в код на мові C, що був розроблений Bell Laboratories. Програма F2C заснована на ядрі першого компілятора Фортрана 77 – f77. Оскільки f77 написаний на мові C і використовує частину компілятора C для фінального кроку компіляції, він і його похідні набагато більш портабельні, ніж компілятори, що генерують машинний код напряму. Інструментальний засіб F2C випущений з відкритим вихідним кодом. Декілька великих бібліотек на Fortran таких як LAPACK були випущені на C з використанням F2C. Також F2C мав певний вплив на створення компілятора GNU g77.

Основною мотивацією використання GCC на противагу F2J/F2C є те, що він охоплює більше діалектів мови Фортран. Сучасна програма на Фортрані може використовувати такі версії мови як 77, 90, 95, 2003, а також поєднання різних версій – всі ці можливості підтримуються компіляторами GCC. З іншого боку, транслятори F2J/F2C працюють лише з програмами на Фортрані 77 і, крім того, оптимізовані лише для певних математичних бібліотек. Також виокремлення абстрактного дерева синтаксису з компілятора GCC є більш простим і потребує лише передачі спеціальних ключів під час запуску компілятора, тоді як засоби F2J/F2C не надають такої можливості і потребують модифікації самих інструментальних засобів.

Для сучасних версій мови Фортран (починаючи з 2003) існують спеціалізовані засоби для синтаксичного аналізу, зокрема Open Fortran Parser [15]. Цей засіб містить формальну граматику мови і орієнтований саме на синтаксичний аналіз, а не на компіляцію/перетворення програм. У попередній роботі [6] автори використовували саме цей засіб для аналізу досить простої програми (алгоритм Гаусса розв'язання системи лінійних рівнянь). Проте виявилось, що для більш складних задач, що містять досить багато старого коду, використання Open Fortran Parser є неможливим. Саме тому в даній роботі були розроблені нові засоби на основі компілятора GCC, який є більш універсальним і може розуміти код на Фортрані 77.

2. Структура компілятора GCC

Як і в більшості портабельних компіляторів, процес компіляції в GCC концептуально може бути розбитий на три фази [7].

1. Для кожної мови, що підтримується GCC, існує окремий front-end. Front-end приймає на вхід програмний код і перетворює його в семантично еквівалентне, незалежне від мови абстрактне дерево синтаксису (AST, Abstract Syntax Tree). Семантика і синтаксис AST визначаються мовою GIMPLE – найвищий незалежний від мови рівень проміжного представлення програми, присутній в GCC.
2. На другій фазі отримане абстрактне дерево синтаксису пропускається через список незалежних від цільової платформи трансформацій коду, які забезпечують такі можливості як конструювання графа потоку керування (CFG, Control Flow Graph), оптимізація AST, генерація проміжного представлення у вигляді нестрогої мови регістрових передач (RTL, Register Transfer Language), крок розширення та запуск оптимізацій, що базуються на RTL. Проміжне представлення у вигляді нестрогої мови RTL передається для подальших низькорівневих проходів компілятора.
3. Низькорівневі проходи компілятора (back-end) є частиною процесу генерації коду. Вони працюють з проміжним представленням, застосовують до нього низькорівневі оптимізації та генерують на його основі асемблерний код для заданої архітектури процесорів. Ці фази також спираються на розширювану множину back-end компонент для кожної цільової архітектури. При цьому набори front-end (що відповідають вхідним мовам) та back-end (що відповідають цільовим архітектурам процесорів) є незалежними, за рахунок чого досягається широка підтримка як вхідних мов, так і цільових архітектур.

У роботі використовується результат першої фази роботи компілятора – абстрактне дерево синтаксису. В системі GCC присутні три формати представлення абстрактного дерева синтаксису:

- ORIGINAL;
- GENERIC;
- GIMPLE.

ORIGINAL є внутрішнім представленням дерева в кожному front-end компоненті. Тому це представлення є специфічним для кожної мови програмування, що підтримується GCC (хоча деякі мови можуть не підтримувати ORIGINAL і зразу генерувати представлення GENERIC). Недоліком цього представлення є відсутність стандартних засобів його отримання з компілятора: для цього буде необхідно вносити зміни в front-end, що відповідає мові Фортран. Тому в даній роботі представлення ORIGINAL не використовується.

Представлення GENERIC є загальним інтерфейсом GCC для створенням front-end для нових мов. Метою GENERIC є незалежний від мови шлях представлення компоненту програми у вигляді дерева (для компоненту використовується термін «функція», позичений з мови С – для Фортрану мова іде про підпрограми). Після того, як front-end представляє програмний код у вигляді GENERIC, з ним можуть працювати всі інші частини компілятора GCC, незалежно від початкової мови програми. Більшість можливостей представлення GENERIC засновані на використанні мови С, проте вони досить точно відображають і інші процедурні мови, зокрема Фортран.

Представлення GIMPLE є підмножиною представлення GENERIC, яке використовується як трьохадресне проміжне представлення коду. Саме з представленням у вигляді GIMPLE працюють всі подальші проходи компілятора. Проте це представлення є досить далеким від коду програми, зокрема в ньому не зберігаються імена змінних.

В даній роботі саме проміжне представлення GENERIC використовується для перетворення програмного коду мовою Fortran у терм, придатний для використання в системі переписувальних правил Termware. GENERIC був вибраний основним робочим проміжним представленням, оскільки це найперше незалежне від мови абстрактне представлення програми, яке можна видобути із GCC. Завдяки цьому абстрактне дерево є найближчим до вихідного коду програми – зокрема, достатньо близьким для подальшого відтворення тексту програми з цього представлення. Це є значною перевагою перед подальшими ще більш абстрактними форматами через необхідність повернення до коду на Фортрані після розпаралелювання. Крім того, розроблені в даній роботі засоби перетворення можуть використовуватись і для інших мов, що підтримуються GCC.

3. Підхід до розпаралелювання з використанням техніки переписувальних правил

У роботі використовується наступний підхід для розробки ефективних паралельних програм для мультиядерних платформ. Вхідними даними є початковий текст послідовної програми мовою Фортран, що описує алгоритм вирішення задачі. Програмний код представляється у вигляді високорівневої моделі з використанням алгебри алгоритмів Глушкова [16]. Далі до цього високорівневого представлення застосовуються перетворення, представлені у вигляді переписувальних правил. Перетворення можуть бути спрямовані на перехід від послідовної до паралельної версії програми (розпаралелюючі перетворення), а також на підвищення продуктивності паралельної програми (оптимізуючі перетворення – в даній роботі вони не використовувались, хоча в подальшому можливе їх використання). Після застосування перетворень до високорівневої моделі програми, використовується генератор коду для створення програмного коду перетвореної програми. Основні етапи процесу розпаралелювання показано на рис. 1.

Багато етапів описаного процесу виконуються в автоматичному або автоматизованому режимі. Так, перехід від початкового тексту програми до її високорівневої алгебраїчної моделі, а також перехід у зворотному напрямку (генерація коду) виконується автоматично, для заданої предметної області і мови програмування (цей процес докладно описаний у розділах 5 та 7). Застосування перетворень програм, представлених у вигляді переписувальних правил, також є автоматизованим: від користувача потрібно лише вказати, які перетворення слід застосовувати і на якій ділянці коду вони повинні діяти. Найбільш складною частиною описаного процесу,

виконуваної вручну, є створення переписувальних правил, що описують перетворення. Проте багато з таких правил застосовні для різних програм, що сприяє їх повторному використанню. При цьому, оскільки правила оперують з високорівневими алгебраїчними моделями, які є незалежними від мови програмування, можливе розпаралелювання програм на Фортрані з використанням перетворень, розроблених раніше для інших мов програмування.

Рис. 1. Основні етапи розпаралелювання

На всіх етапах розпаралелювання використовується техніка переписувальних правил, реалізована системою Termware [8, 9]. Система Termware спрямована на розробку високодинамічних прикладних систем, до яких пред'являються підвищені вимоги, як до вбудованого інтелекту для забезпечення інтерактивності розроблювальних систем, так і до здешевлення розробки, скорочення строків проектування і поліпшення характеристик повторного використання і супроводжуваності. Вона відрізняється від більшості інших систем цього класу як призначенням і семантикою використовуваних засобів, так і технологією їх реалізації. Мова Termware не є універсальною мовою програмування в тому розумінні, що вона не призначена для написання повнофункціональних програмних систем. Це координаційна мова-оболонка, призначена для написання предметно-орієнтованих частин застосувань, що вбудовуються в прикладну систему для реалізації функцій взаємодії цієї системи із програмним оточенням.

Для системи Termware характерні наступні риси:

- використання декларативного опису;
- реалізація тільки необхідної частини застосування, а не всього застосування;
- орієнтація на тісну взаємодію з іншими частинами застосування.

Переписувальні правила описуються з використанням синтаксису мови Termware і зберігаються в окремих файлах. Основою мови є терми, тобто вирази виду $f(x_1,...,x_n)$. Атомарними термами є символи-змінні (які записуються у вигляді *$var*), а також константи певних типів даних (числових, логічних, текстових і атомарного – незмінні рядки). Для спрощення запису і сприйняття використовуються скорочення для багатьох термів, наприклад *x+y* – для *plus(x,y)*; [*x:y*] – для *cons*(*x,y*) (список з першим елементом x і залишком списку y); *x ? y: z* – для *ifelse*(*x,y,z*) і інші. Увесь набір правил є термом, записаним з використанням цих скорочень.

Правила Termware мають наступний загальний вигляд:

source [condition] -> destination [action]

Тут використовуються чотири терми:

- *source* – вхідний зразок;
- *destination* – вихідний зразок;
- *condition* – умова, що визначає застосовність правила;
- *action* – дія, виконувана при спрацьовуванні правила.

Виконувані дії і умови, що перевіряються є необов'язковими компонентами правила, які можуть викликати імперативний код. За рахунок цього базова модель переписування термів може бути розширена довільними додатковими можливостями.

4. Побудова алгебраїчної моделі програми з використанням GCC та Termware

Запропонований підхід до розпаралелювання програми на мові Фортран передбачає представлення коду програми у вигляді високорівневої алгебраїчної моделі. Побудова такої моделі проходить у декілька етапів. Першим з них є лексичний і синтаксичний аналіз програмного коду і побудова абстрактного дерева синтаксису. Власне для цього і було задіяно GCC.

Серед вхідних параметрів компілятора GCC є такі, що дозволяють видобути текстове представлення абстрактного дерева синтаксису (у формі GENERIC) під час компіляції. У залежності від переданого параметру

є можливість вибрати, на якому етапі компіляції буде надруковано дерево, що дозволяє при потребі надрукувати дерево у чистому вигляді без додаткових перетворень компілятора.

Команда **gcc --fdump-tree-original-raw** над файлом із програмою на Фортрані видає на виході дерево у наступному форматі:

```
exits (integer(kind=4) & restrict k)
@1    function_decl   name: @2      mngl: @3      type: @4
                      scpe: @5      srcp: ElDen.for:1294
                      args: @6      link: extern  body: @7
@2    identifier_node strg: exits   lngt: 5
@3    identifier_node strg: exits_  lngt: 6
@4    function_type   size: @8      algn: 8       retn: @9
                      prms: @10
@5    translation_unit_decl
@6    parm_decl       name: @11     type: @12     scpe: @1
                      srcp: ElDen.for:1294        argt: @12
                      size: @13     algn: 64      used: 1
@7    bind_expr       type: @9      vars: @14     body: @15

…
```

Даний текст є фактично направленим графом, кожен з вузлів якого складається з:

- порядкового номеру (наприклад, @1), який використовується для ідентифікації вузла в дереві;
- імені вузла (наприклад, function_decl);
- атрибутів вузла (наприклад, name: @2 або lngt: 5).

Значеннями атрибутів можуть бути константи (текстові або числові), а також посилання на інші вузли, які задають дуги графа.

Насправді побудований граф ще не є деревом синтаксису, оскільки містить циклічні залежності між вузлами. Наприклад, можна бачити, що вузол @1 має посилання args: @6, тоді як вузол @6 має зворотне посилання scpe: @1. Тому перетворення графа у дерево є додатковим етапом перетворення програми.

Основними технічними задачами цього етапу були наступні:

- перетворити вихідний текст дерева синтаксису з формату GCC у вигляд термів, які можуть використовуватись системою Termware;
- позбутись циклічних залежностей між вузлами графа.

Обидві задачі вирішено текстовими перетвореннями над вихідним текстом GCC компілятора, з використанням засобів awk та мови PHP.

Для процесу видалення циклів використовується наступний алгоритм. Побудуємо граф, вузли якого відповідатимуть вузлам синтаксичного дерева, а наявність ребер вказуватиме на наявність синтаксичної залежності між вузлами. Для видалення циклів застосуємо обхід графа в ширину, починаючи з вершини, що відповідає визначенню підпрограми (вона завжди має порядковий номер @1) і помічаючи кожну пройдену вершину. Якщо чергова вершина містіть перехід на вже відмічену вершину, то застосовується процедура видалення циклу. Процедура аналізує тип вхідної і вихідної вершин і вирішує, який з переходів має бути видалено. Наприклад, серед популярних циклів є посилання із вершини, що відповідає за декларацію змінної, у вершину що відповідає за тип змінної, і одночасно зворотне посилання на попередню вершину, як ту, яка використовує вказаний тип. У цьому випадку зворотне посилання несуттєве для аналізу і буде вилучено.

Крім видалення циклічних посилань, на даному етапі також видаляються деякі атрибути, які використовуються для подальшої роботи компілятора GCC, проте непотрібні для створення алгебраїчної моделі. Прикладами таких атрибутів є type (для підпрограм він є пустим, але відображається у вигляді досить складного виразу), а також scpe, srcp, link (службові атрибути, необхідні для роботи подальших фаз компіляції).

Після видалення циклів, отримане дерево обходу графа в ширину вже можна вважати абстрактним деревом синтаксису. Воно перетворюється до вигляду термів наступним чином: кожний вузол з іменем A стає термом з іменем A та підтермами, що відповідають атрибутам. Кожен атрибут з іменем B стає термом з іменем B та єдиним підтермом, що відповідає значенню атрибута. Приклад вхідної частини графа і вихідного терму отриманого із нього, наведено в табл. 1.

Отриманий терм розглядається як низькорівнева синтаксична модель тексту програми. Наступним етапом є перетворення цієї синтаксичної моделі у високорівневу алгебраїчну модель з використанням переписувальних правил спеціального вигляду – патернів [3]. У загальному випадку, патерн визначається двома системами правил: R_p – система правил для виділення патерна з довільного терма, R_g – система правил для розшифрування патерна. У більш частковому випадку патерн задається парою термів t_p – позначення патерна (елемент моделі високого рівня) і t_g – зразок, що задає патерн (елемент моделі низького рівня). У цьому випадку $R_p = \{t_g \to t_p\}$ і $R_g = \{t_p \to t_g\}$.

Такого роду патерни задаються для всіх елементів високорівневої моделі t_p і задають їх представлення у вигляді елементів низькорівневої моделі t_g . Наприклад, можна визначити патерн для параметрів функції

t_p = Param($name),

t_g = parm_decl($id, name(identifier_node(strg($name),lngt($l))), argt($ar), algn($al), used($us))

Таблиця 1. Перетворення графа в дерево (терм)

Вузли вхідного графа	**Вираз у вигляді терму**
exits (integer(kind=4) & restrict k) @1 function_decl name: @2 mngl: @3 type: @4 scpe: @5 srcp: ElDen.for:1294 args: @6 link: extern body: @7 @2 identifier_node strg: exits lngt: 5 @3 identifier_node strg: exits_ lngt: 6 @6 parm_decl name: @11 type: @12 scpe: @1 srcp: ElDen.for:1294 argt: @12 size: @13 algn: 64 used: 1 @7 bind_expr type: @9 vars: @14 body: @15	function_decl (identifier(node1), name(identifier_node (strg(exits), lngt(5))), mngl(identifier_node (strg(exits_), lngt(6))), type(), args(parm_decl (identifier(node6), name(identifier_node (strg(k), lngt(1))), argt(2), algn(64), used(1))), body(bind_expr (...)))

Використання переписувальних правил у вигляді патернів дозволяє значно зменшити розмір моделі. Крім цього, можна позбутись залежності від конкретного формату абстрактного дерева синтаксису, який є специфічним для кожного синтаксичного аналізатора. Наприклад, у [6] використано синтаксичний аналізатор мови Фортран на основі інструментального засобу Open Fortran Parser, тоді як в даній роботі використано аналізатор, який є компонентом GCC. Тому низькорівневі синтаксичні моделі для цих аналізаторів є несумісними. Проте високорівневі алгебраїчні моделі можуть бути приведені до однакового вигляду за рахунок використання відповідних патернів, що дозволяє використати перетворення, створені для Open Fortran Parser, для обробки програм, які були оброблені з використанням GCC.

5. Розпаралелювання з використанням системи Termware

Після перетворення тексту програми в алгебраїчну модель можна застосовувати перетворення, спрямовані на розпаралелювання та оптимізацію паралельної програми. В роботі розглядаються перетворення, орієнтовані на розпаралелювання програми на Фортрані з використанням засобів OpenMP. Зокрема застосовуються перетворення розпаралелювання циклів з незалежними ітераціями.

Такі перетворення вже розглядались у [6], але там розпаралелювалась більш проста задача (алгоритм Гаусса вирішення систем лінійних рівнянь). В даній роботі розглядається реальна прикладна задача, хоча і не дуже велика за обсягом. Оскільки алгоритм обчислень не є очевидним, розпаралелювання потребує використання засобів автоматизації.

Першим етапом розпаралелювання є вибір циклів, які можна перетворити на паралельні. На відміну від більш простих задач (зокрема, алгоритму Гаусса), цей етап є досить складним. В початковому тексті програми містилось 54 цикли, які відповідали як за обчислення, так і за технічні особливості реалізації. Пошук найбільш придатного для розпаралелювання циклу можна було б виконати вручну, але це потребує знань алгоритму та досить великих витрат часу. Тому в даній роботі використано засоби автоматизації, а саме профілювальник Intel VTune Amplifier [12] в поєднанні з технікою переписувальних правил.

Інструментальний засіб Intel VTune Amplifier входить до складу системи Intel Parallel Studio та являє собою розвиток попередніх інструментальних засобів Intel Thread Profiler та VTune Performance Analyzer. Він дозволяє знаходити фрагменти коду, які виконуються найчастіше, і таким чином є першими кандидатами для розпаралелювання. (Іншою можливістю Intel VTune Amplifier є аналіз продуктивності мультипоточних програм

та пошук недостатньо ефективних конструкцій синхронізації. В попередній роботі [5] вже використовувались засоби Intel Thread Profiler для такого аналізу в поєднанні з системою переписувальних правил Termware для автоматизації розпаралелюючих та оптимізуючих перетворень програми на C++. Проте в даній роботі такі можливості Intel VTune Amplifier не використовуються).

Застосування Intel VTune Amplifier дозволило знайти фрагмент програми, що виконується найчастіше (див. рис. 2). Проте перетворення розпаралелювання може застосовуватись не безпосередньо до даного фрагменту, а скоріше до одного з циклів, що містить цей фрагмент. Саме такі цикли є першими кандидатами на застосування розпаралелюючого перетворення, оскільки ефект від їх розпаралелювання буде максимальним. Проте пошук цих циклів може бути досить складною задачею через великий обсяг коду та за умови недостатнього розуміння сутності алгоритму. Але такий пошук може бути автоматизований з використанням засобів переписувальних правил Termware. Для цього користувач помічає знайдений за допомогою Intel VTune Amplifier фрагмент за допомогою терму-мітку _MarkHotspot. Після цього використовується наступна система правил:

1) [_MarkHotspot($x):$y] -> _MarkHotspot([$x:$y]);
2) [$x:_MarkHotspot($y)] -> _MarkHotspot([$x:$y]);
3) DoCnt($var,$start,$end, _MarkHotspot($body)] ->
_MarkHotspot(DoCnt($var,$start,$end,$body,_MARK_CAND_PARALLEL));
4) Function($name, $params, $return,_MarkHotspot($body)) ->
Function($name,$params,$return,$body,_MARK_PASSED) [addItem($name)];
5) Call($name,$params) [hasItem($name)] ->_MarkHotspot(Call($name,$params)).

Ця система працює наступним чином. Правила 1) та 2) розповсюджують мітку на сусідні фрагменти в рамках одного циклу. Правило 3) спрацьовує, коли все тіло циклу помічене міткою _MarkHotspot; це правило додає до циклу мітку _MARK_CAND_PARALLEL, а мітка _MarkHotspot піднімається на вищий рівень. Правила 4) та 5) потрібні для пошуку подібних циклів по всім функціям (підпрограмам), які викликають даний фрагмент коду. Правило 4) запам'ятовує імена всіх таких функцій (і помічає їх міткою _MARK_PASSED, що забороняє їх повторну обробку). Правило 5) знаходить всі виклики подібних функцій і помічає їх міткою _MarkHotspot, щоб далі до них могли застосовуватись правила 1)–3).

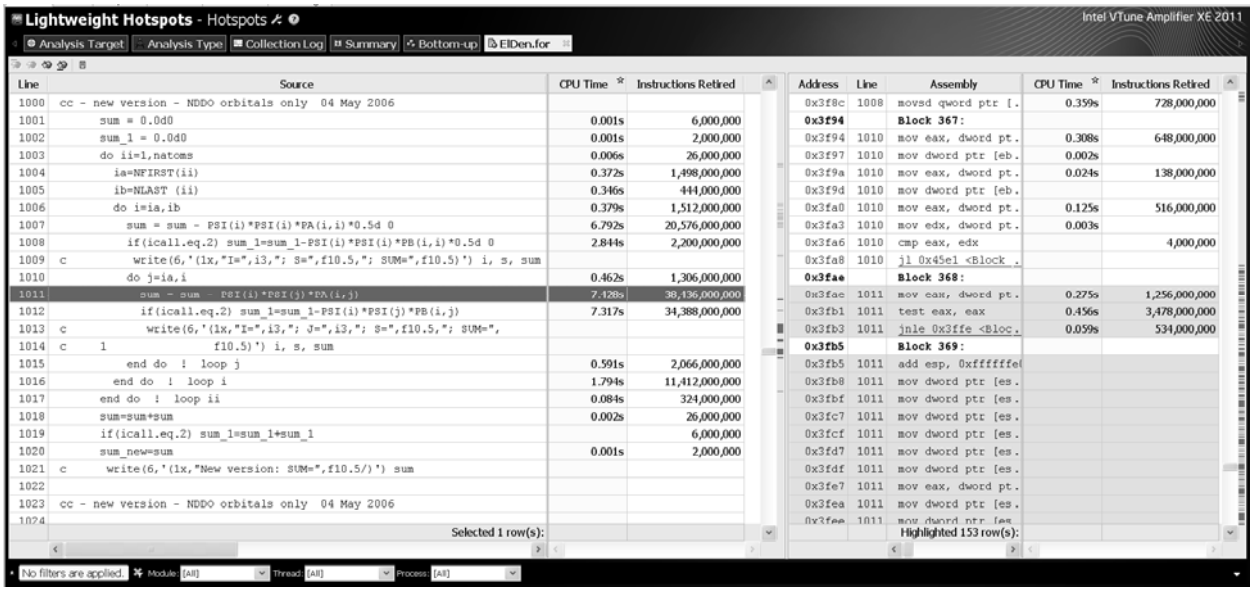

Рис. 2. Знаходження фрагменту коду для розпаралелювання за допомогою Intel VTune Amplifier

Після спрацьовування даної системи правил всі цикли, з яких викликається найбільш використаний фрагмент коду, будуть помічені міткою _MARK_CAND_PARALLEL. В програмі розрахунку електронної густини таких циклів виявилось 6, і саме їх слід розглядати в якості кандидатів для розпаралелювання. Таким чином, використання інструментальних засобів Intel VTune Amplifier та системи переписувальних правил Termware дозволило скоротити кількість кандидатів на розпаралелювання з 54 до 6, значно спростивши процес розпаралелювання.

На наступному етапі розпаралелювання необхідно вибрати один з 6 вкладених циклів, до якого буде застосовано перетворення. Вибір можна проводити, користуючись наступними міркуваннями. З одного боку, вибір більш глибоко вкладених циклів призводить до збільшення накладних витрат на розпаралелювання, оскільки кожний раз, коли виконується такий цикл, потрібно буде розбити ітерації по потокам та очікувати завершення всіх потоків. З іншого боку, вибір зовнішніх циклів може бути неможливим, оскільки їх ітерації є залежними, або потребують великої кількості синхронізацій для забезпечення коректної роботи зі спільною пам'яттю. Крім того, зовнішні цикли можуть не мати достатньої кількості ітерацій, щоб завантажити всі доступні ядра процесору.

З огляду на наведені міркування, для розпаралелювання логічно обрати найбільш зовнішній цикл, який може бути розпаралелений. В задачі обчисленні електронної густини зовнішній цикл не містив достатньо ітерацій, тому було обрано другий цикл ззовні. Після вибору такого циклу він помічається міткою _MARK_CAND_PARALLEL.

Наступним етапом є застосування розпаралелюючого перетворення. Перетворення є досить простим і повністю відповідає описаному в роботі [6] для іншої задачі. Перетворення реалізується наступним правилом:

DoCnt($var,$start,$end,$body,_MARK_Parallel)->ParallelDoCnt($var,$start,$end,$body)

Елемент моделі ParallelDoCnt (паралельний цикл) далі перетворюється в реалізацію з використанням директив OpenMP, як описано в роботі [6].

6. Генерація тексту паралельної програми на мові Фортран

Останнім етапом розпаралелювання є генерація тексту паралельної програми на Фортрані з перетвореної алгебраїчної моделі. Генерація виконується у зворотному порядку порівняно до синтаксичного аналізу: спочатку алгебраїчна модель перетворюється на синтаксичну, потім з синтаксичної моделі відтворюється текст програми. Перший етап генерації виконується за допомогою патернів, які виконуються в зворотному напрямку (система R_g).

Наступний етап – генерація коду мовою Фортран з терму (низькорівневої синтаксичної моделі) паралельної програми. Паралельні конструкції терму представляються на мові Фортран з використанням директив OpenMP.

Генерація коду поєднує в собі текстовий аналіз терму, побудову дерева розбору терму, генерацію тексту програми на основі вузлів дерева. Оскільки вузли дерева синтаксично залишаються тими самими, що були отримані із вхідної програми на мові фортран, то вхідна програма може бути відтворена з точністю до назв змінних у програмі.

Перетворення терму у програмний код проходить у три етапи.

1. На першому кроці терм розбивається на окремі лексеми за набором символів, до яких входять дужки і кома.
2. На другому кроці запускається процедура, яка аналізує, який символ знаходився між двома лексемами і тип лексеми. На основі цієї інформації процедура приймає рішення про місце вершини для наступної лексеми у дереві: чи вона буде нащадком поточної вершини або рекурсивно поточна вершина ставатиме батьківською для всіх наступних поки не з'явиться ознака повернення із рекурсії. Такою ознакою є вміст поточної лексеми: якщо вона є порожнім рядком, тобто між двома сусідніми дужками не було лексем, то це є ознакою повернення на один рівень дерева вгору. Після закінчення другого кроку ми отримуємо синтаксичне дерево програми.
3. На третьому кроці будується код програми на основі синтаксичного дерева. Кожна гілка дерева містить інформацію, яка використовуватиметься для генерації коду. В табл. 2 наведено відповідності між синтаксичними вузлами дерева GCC і лексемами, що мають будуватись на мові Фортран.

Таблиця 2. Генерація коду на Фортрані

Елемент синтаксичного дерева	**Значення**
function_decl	декларація функції, subroutine
name(%nodeName%)	ідентифікатор функції або змінної %nodeName%
bind_expr	тіло функції або циклу
body(statement_list(%1%, %2%))	набір операцій що належать до тіла функції або циклу
modify_expr	бінарний оператор присвоєння
loop_expr	оператор циклу
var_decl	декларація змінної
plus_expr	бінарний оператор додавання
float_expr	унарний оператор приведення до типу float
integer_cst	константа цілого типу
cond_expr	тернарний оператор розгалудження
array_ref	доступ до елементу масиву
call_expr	виклик функції

Генерація тексту паралельної програми на тій самій мові, на якій було написано послідовну програму, дозволяє спеціалістам в предметній області орієнтуватись в паралельній програмі і вносити зміни, що не стосуються паралельної частини. Крім цього, в перетвореному коді явно вказано, які саме фрагменти було розпаралелено. Це є перевагою порівняно з використанням розпаралелюючих компіляторів, де користувач має дізнаватись про розпаралелені фрагменти програми зі спеціальних звітів, згенерованих компіляторами.

7. Приклад розпаралелювання: задача обчислення електронної та спінової густини

У роботі підхід до розпаралелювання програм на Фортрані проілюстровано на прикладній задачі з галузі квантової хімії. Програма обчислює електронну та спінову густину атомів поліциклічних ароматичних

вуглеводнів на прямокутній сітці заданого розміру *N*. Квантово-хімічні розрахунки виконуються за допомогою напівемпіричних та неемпіричних методів [11]. Програма має також візуалізаційну частину, що проводить візуалізацію квантово-хімічного експерименту, який являє собою математичну модель поведінки атомів у молекулі. В даній роботі розглядається лише обчислювальна частина програми.

Код програми (обчислювальної частини) представлений у вигляді одного файлу ElDen.for, в якому містяться всі компоненти програми – власне алгоритм обчислень, вивід результатів та допоміжних даних, а також частина вхідних даних, наприклад, параметри електронних оболонок для основних хімічних елементів. Обсяг програми складає 1680 рядків коду. В коді програми присутні конструкції, характерні для різних версій мови Фортран – від Фортрану 77 до Фортрану 95, а також нестандартні розширення (деякі з них, зокрема функції GETTIM та GETDAT, довелось видалити з коду, оскільки вони перешкоджали роботі компілятора GCC). Програма була розроблена для компіляції з використанням Compaq Visual Fortran 6.6, хоча з незначними модифікаціями може бути скомпільована сучасними комерційними компіляторами Intel Fortran Composer [12] та PGI Fortran Compiler [13].

В роботі [10] розглядалось автоматичне розпаралелювання даної програми з використанням компіляторів Intel Fortran Composer та PGI Fortran Compiler. Відмінність даної роботи полягає в використанні перетворень на рівні тексту програми. Як видно з подальших результатів вимірювання часу роботи, такі перетворення досягають такого ж рівня ефективності, як і комерційні компілятори.

Для оцінки ефективності розпаралелювання було проведено вимірювання часу роботи послідовної (SEQ) та паралельної (PAR) програм. Вимірювання проводилось для різних розмірів задачі *N* (деталізація сітки, від 200 до 800) на чотириядерному процесорі Intel Core2 Quad Q8200. Результати вимірювань показано на рис. 3.

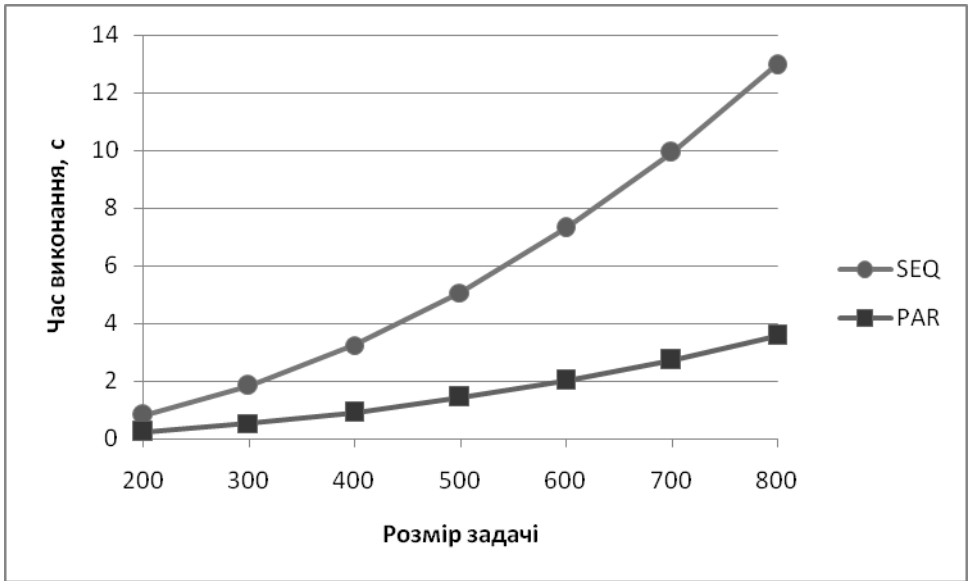

Рис. 3. Порівняння часу виконання послідовної та паралельної версії

Як видно з результатів вимірювання, розпаралелювання призводить до значного збільшення продуктивності. В залежності від розміру задачі, було досягнуто прискорення від 3,3 до 3,6 разів порівняно з послідовною програмою (для менших розмірів задачі прискорення виявляється меншим, оскільки більшого значення набувають накладні витрати на синхронізацію потоків). Ефективність розпаралелювання з використанням техніки переписувальних правил може бути порівняна з результатами роботи автоматичного розпаралелювання компіляторів Intel Fortran Composer та PGI Fortran Compiler: в [10] для цієї ж задачі використання розпаралелюючих компіляторів дало змогу досягти прискорення в 6 разів на 8 обчислювальних ядрах, що співвідноситься з результатами даної роботи.

Зазначимо, що в порівнянні з [10], програма в обох варіантах (послідовному та паралельному) в даній роботі працює значно швидше. Це пов’язано з видаленням фрагменту коду, який не впливає на результат, але обчислювався в попередній версії програми. Видалення такого фрагменту значно підвищило ефективність програми: для розміру задачі N=200 час виконання попереднього варіанту програми складав 7,6 с, тоді як нового варіанту 0,82 с. Даний фрагмент було знайдено вручну, під час аналізу результатів, отриманих за допомогою Intel VTune Amplifier (хоча представлення програми у вигляді алгебраїчної моделі спростило аналіз). В подальшому автори планують автоматизувати пошук подібних фрагментів коду з використанням техніки переписувальних правил, оскільки вони можуть досить часто зустрічатись в застарілому коді.

Висновки

В роботі описано підхід до розпаралелювання програм на Фортрані з використанням техніки переписувальних правил. Особливістю даної роботи є використання для синтаксичного аналізу компілятора GCC. Це дозволяє аналізувати застарілий код, що містить суміш різних версій мови Фортран і через це є

важким для синтаксичного аналізу. Підхід застосовано до розпаралелювання прикладної задачі з галузі квантової хімії – програми обчислення електронної та спінової густини. Використання запропонованого підходу дозволило суттєво збільшити ефективність програми за часом виконання.

До подальших напрямків досліджень належить розробка нових переписувальних правил, що описують розпаралелюючі та оптимізуючі перетворення програм, а також аналізують код з метою пошуку помилок. Запропонований підхід планується застосовувати для підвищення ефективності інших прикладних задач, як у галузі квантової хімії, так і в інших галузях. Крім використання засобів OpenMP для підтримки мультиядерних платформ, можливо також автоматизоване використання інших засобів програмування для платформ з розподіленою пам'яттю, графічних прискорювачів, грід та хмарних обчислень. Також можлива адаптація розроблених засобів для інших мов, що підтримуються компілятором GCC.

1. *Krste Asanovic, et al.* The Landscape of Parallel Computing Research: A View from Berkeley, Electrical Engineering and Computer Sciences, University of California at Berkeley, Technical Report No. UCB/EECS-2006-183, December 18, 2006. http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.html
2. *Buttari A., Dongarra J., Kurzak J., et al.* The impact of multicore on math software. In B. Kagstrom, E. Elmroth, J. Dongarra, and J.Wasniewski, editors, Applied Parallel Computing. State of the Art in Scientific Computing, PARA 2006, Umea, Sweden, June 2006. LNCS 4699. – 2007. – P. 1–10.
3. *Андон Ф.И., Дорошенко А.Е., Жереб К.А.* Программирование высокопроизводительных параллельных вычислений: формальные модели и графические ускорители // Кибернетика и системный анализ. – 2011. – № 4. – С. 176–187.
4. *Дорошенко А.Е., Жереб К.А.* Техника и инструментарий переписывающих правил для инженерии программного обеспечения графических ускорителей // Инженерия программного обеспечения. – 2010. – № 4. – С. 35–49.
5. *Дорошенко А.Е., Жереб К.А., Яценко Е.А.* Формализованное проектирование эффективных многопоточных программ // Проблеми програмування. – 2007. – № 1. – С. 17–30.
6. *Дорошенко А.Е., Жереб К.А., Тырчак Ю.М., и др.* Создание эффективных параллельных программ на Фортране с использованием техники переписывающих правил. Материалы конференции "Высокопродуктивные вычисления" (HPC-UA'2011). Киев, 12–14 октября 2011 г. – С. 76–83.
7. *Richard M. Stallman.* Using the Gnu Compiler Collection: A Gnu Manual for Gcc Version 4.3.3. CreateSpace, Paramount, CA. – 2009. – 636 p.
8. *Doroshenko A., Shevchenko R.* A Rewriting Framework for Rule-Based Programming Dynamic Applications. Fundamenta Informaticae. – 2006.– Vol. 72, N 1–3.– P. 95–108.
9. *TermWare*. – http://www.gradsoft.com.ua/products/termware_rus.html
10. *Дорошенко А.Ю., Хаврюченко В.Д., Єгоров В.І., та ін.* Паралельна реалізація алгоритму для задачі розрахунку густини молекулярних систем // Тези доповідей міжнародної конференції TAAPSD'2011 «Теоретичні та прикладні аспекти побудови програмних систем» (Україна, Ялта, 18-23 вересня, 2011 р.), Кримський гуманітарний університет, 2011. – С. 75–77.
11. *Khavryutchenko V.D., Tarasenko Y.A., Strelko V.V., et al.* Quantum chemical study of polyaromatic hydrocarbons in high multiplicity states // International Journal of Modern Physics B. – 2007.– Vol. 21, N 26. – P. 4507–4515.
12. *Intel* Parallel Studio http://software.intel.com/ru-ru/articles/intel-parallel-studio/
13. *Portland* Group Products http://www.pgroup.com/products/index.htm
14. *Fortran* Compilers: Gnu Compiler Collection, IBM VisualAge, Intel Fortran Compiler, Low Level Virtual Machine, Open64, PathScale, GFortran, F2C. Books Nippan. – 2010. – 48 p.
15. *Open* Fortran Parser (OFP) http://fortran-parser.sourceforge.net/
16. *Андон Ф.И., Дорошенко А.Е., Цейтлин Г.Е., Яценко Е.А.* Алгеброалгоритмические модели и методы параллельного программирования. – Киев: Академпериодика, 2007. – 631 с.