УДК 004.75

А.С. ДОВБИШ, О.Б. БАРИЛО

Сумський державний університет, Україна

ОСОБЛИВОСТІ ПРОЕКТУВАННЯ БАГАТОПОТОЧНИХ ПРОГРАМ

Розглянуто підхід до розроблення багатопоточних програм шляхом розпаралелюванняпослідовних алгоритмів та розбиття послідовної програми на підзадачі в залежності від кількості вузлів обчислень та об'єктів дослідження. Проаналізовано випадки розпаралелювання послідовних алгоритмів, при яких кількість потоків дорівнює кількості вузлів обчислення та кількості об'єктів дослідження. На прикладі задачі сортування масивів показано перевагу по оперативності реалізації паралельних алгоритмів обчислень у порівнянні з послідовними. При цьому встановлено, що використання розпаралелювання, при якому кількість потоків дорівнює кількості файлів обробки є найдоцільнішим як з точки зору проектування та реалізації програми, так і за швидкодією.

Ключові слова: *програмне забезпечення, алгоритм, підзадача, потік, вузол обчислення, паралельна програма, послідовна програма, ефективність.*

Вступ

Гостра необхідність у підвищенні продуктивності програмного забезпечення для вирішення трудомістких завдань, з одного боку, і нові можливості розпаралелювання обчислень, що надаються багатоядерними архітектурами сучасних мікропроцесорів – з іншого, спонукає до створення спеціалізованого інструментарію для розробки паралельних програм для таких багатоядерних архітектур.

Головним шляхом підвищення продуктивності програм для таких платформ є розпаралелювання програм з використанням багатопоточності [1]. Проблема полягає не тільки в необхідності створення інструментів для проектування нових програм, але також в забезпеченні можливості реінженерії існуючого програмного забезпечення при його перенесення з застарілих на нові багатоядерні платформи.

Розробка ефективних програм для багатоядерних архітектур є масштабною науково-технічною проблемою, успішне вирішення якої може бути забезпечене тільки шляхом спеціалізації на предметні області і глибокого охоплення етапів життєвого циклу розроблюваних програм із застосуванням засобів автоматизації проектування та програмування, від написання первинних специфікацій до генерації виконуваного коду.

Основою для такої автоматизації є, перш за все, високорівнева формалізація конструювання багатопоточних програм та автоматизація формальних трансформацій програм з метою оптимізації їх продуктивності. Під оптимізацією програм розуміється досягнення необхідних якісних характеристик програми за заданими критеріями (пам'ять, швидкодія, завантаження устаткування та ін.)

Паралельні обчислення використовувалися багато років в основному в найпродуктивніших обчисленнях, але останнім часом до них зріс інтерес внаслідок існування фізичних обмежень на зростання тактової частоти процесорів.

Поява в останні роки великої кількості порівняно дешевих кластерних паралельних обчислювальних систем призвело до швидкого розвитку паралельних обчислювальних технологій, у тому числі і в області високопродуктивних обчислень. Останнім часом ситуація в галузі паралельних обчислювальних технологій почала кардинально змінюватися в зв'язку з тим, що більшість основних виробників мікропроцесорів стали переходити на багатоядерні архітектури.

Зміна апаратної бази змушує змінювати і підхід до побудови паралельних алгоритмів. Для реалізації обчислювальних можливостей багатоядерних архітектур потрібна розробка нових паралельних алгоритмів. Ефективність використання обчислювальних ресурсів, яку будуть мати кластери нового покоління, напряму залежатиме від якості власне паралельних програм, так і спеціалізованих бібліотек чисельних алгоритмів, орієнтованих на багатоядерні архітектури.

У статті розглянуто підходи до розроблення багатопоточних програм шляхом розпаралелення послідовних алгоритмів та розбиття послідовної програми на під задачі в залежності від кількості вузлів обчислення та об'єктів дослідження. На прикладі задачі сортування масивів було продемонстровано різницю в швидкодії між послідовною та паралельною програмами та показано завантаженість устаткування.

Особливості розроблення багатопоточних програм

Паралельні обчислювальні системи — це фізичні комп'ютерні, а також програмні системи, що реалізують той чи інший спосіб паралельну обробку даних на багатьох обчислювальних вузлах.

Ідея розпаралелювання обчислень заснована на тому, що більшість задач може бути розділене на набір менших задач, які можуть бути вирішені одночасно.

Зазвичай паралельні обчислення потребують координації дій. Паралельні обчислення існують в декількох формах: паралелізм на рівні бітів, паралелізм на рівні інструкцій, паралелізм даних, паралелізм завдань. Паралельні обчислення використовувалися багато років в основному у високопродуктивних обчисленнях, але останнім часом до них зріс інтерес внаслідок існування фізичних обмежень на зростання тактової частоти процесорів. Паралельні обчислення стали домінуючою парадигмою в архітектурі комп'ютерів, в основному у формі багатоядерних процесорів [1].

Обчислення називають паралельними, якщо в один і той же момент часу одночасно виконується кілька операцій обробки даних [2]. Можна прискорити процес вирішення задачі за допомогою поділу алгоритму на деякі незалежні частини, які будуть виконуватися на окремих процесорах. Однак паралелізм ще не отримав широкого розповсюдження. Це зумовлено наступними факторами [3]:

– висока вартість паралельних обчислювальних систем [2];

– втрати продуктивності для організації паралелізму;

– постійне вдосконалення послідовних комп'ютерів - відповідно до закону Мура потужність послідовних процесорів зростає практично у два рази кожні 18-24 місяців;

– існування послідовних обчислень;

– залежність ефективності паралелізму від обліку характерних властивостей паралельних систем;

– існуюче програмне забезпечення орієнтовано в основному на послідовні ЕОМ [4].

Писати програми для паралельних систем складніше, ніж для послідовних [5], так як конкуренція за ресурси представляє новий клас потенційних помилок в програмному забезпеченні, серед яких стан перегонів є найпоширенішою. Взаємодія і синхронізація між процесами представляють велику перешкоду для отримання високої продуктивності паралельних систем. В останні роки також стали розглядати питання про споживання електроенергії паралельними комп'ютерами .

Характер збільшення швидкості програми в результаті розпаралелювання пояснюється законом Амдала, який ілюструє обмеження зростання продуктивності обчислювальної системи зі збільшенням кількості обчислювачів. Згідно з цим законом, прискорення виконання програми за рахунок розпаралелювання її інструкцій на безлічі обчислювачів обмежена часом, необхідним для виконання її послідовних інструкцій.

Якщо при обчисленні не застосовуються циклічні (що повторюються) дії, то N обчислювальних модулів ніколи не виконають роботи в N разів швидше, ніж один єдиний обчислювальний модуль.

Наприклад, для швидкого сортування масиву на двопроцесорній машині можна розділити масив навпіл і сортувати кожну половину на окремому процесорі. Сортування кожної половини може зайняти різний час, тому необхідна синхронізація [6].

Розглянемо види розпаралелювання послідовних алгоритмів в залежності від розв'язуваної задачі:

Розпаралелювання послідовного алгоритму. Даний вид розпаралелювання використовується, коли необхідно збільшити швидкість розв'язання конкретної задачі при неможливості чітко виділити підзадачі. Прикладом може бути добуток матриць великої розмірності, сортування масиву, підрахунок кількості простих чисел для заданого числа та інші. [7]

Розпаралелювання не самого алгоритму, а відокремлення його виконання в окремий потік. Даний вид розпаралелювання використовується, коли необхідно збільшити швидкість розв'язання задачі, яку чітко можна розбити на підзадачі. Прикладом може бути обчислення добутку кількох матриць великої розмірності, сортування декількох масивів. Для СКБД доцільно кожен запит користувача обробляти в окремому потоці. Сьогодні сучасні Інтернет браузери розпаралелюють виконання кожної вкладки в окремий потік.

Не доцільно виділяти будь-який з видів розпаралелювання, так як кожен призначений для вирішення проблеми швидкодії задач свого класу. Кожен з випадків є затребуваний на ринку інформаційних технологій, так як постійно збільшується кількість інформації, яку необхідно обробляти (сортувати, групувати, аналізувати і т.д.), що в свою чергу ставить нові завдання перед СКБД та іншим програмним забезпеченням.

Для обох випадків розрізняють розпаралелювання послідовного алгоритму, при якому:

– кількість потоків дорівнює кількості вузлів обчислення (процесорів, ядер). Для даного випадку процес розроблення і реалізація програми достатньо громіткий, так як потребує додатково врахування

параметру «кількість вузлів обчислення» та динамічного розподілення операцій на задану кількість потоків; перевагою є рівномірне навантаження на вузли обчислення, що при великих обчисленнях може значно пришвидшити загальний час роботи програми;

– створюється довільна кількість потоків. Даний випадок є значно простіший в реалізації. Розробник самостійно вирішує на яку кількість потоків необхідно розділити алгоритм, в залежності від складності реалізації і можливостей розпаралелювання алгоритму (при сортуванні масив доцільно розділяти на невелику кількість частин, так як необхідна тісна синхронізація, при добутку матриць доцільно добуток кожного рядка винести в окремий потік); недостатком є в деяких випадках нерівномірне навантаження вузлів обчислення (масив сортується двома потоками на чотирьох ядерному процесорі).

Приклад реалізації багатопоточної програми

Розглянемо на прикладі задачі сортування декількох масивів залежність швидкості сортування від способу організації багатопоточності:

- сортування одним потоком;
- сортування, при якому кількість потоків дорівнює кількості вузлів обчислення;
- сортування, при якому кількість потоків дорівнює кількості масивів, які необхідно відсортувати.

Вхідними даними є N файлів з масивами дійсних чисел.

Сортування одним потоком. Алгоритм сортування N файлів одним потоком виглядає таким чином:

1. Визначається кількість файлів для оброблення: N.
2. Установлюється лічильник I на першому файлі I = 0.
3. Запам'ятовується мітку часу T0.
4. Запам'ятовується мітку часу T1.
5. Сортується I-й файл.
6. Запам'ятовується мітку часу T2.
7. Обчислюється час, затрачений на сортування I-го файлу: T2 – T1.
8. Збільшується лічильник I = I+1.
9. Якщо лічильник I менше кількості файлів N – виконується крок 4.
10. Запам'ятовується мітку часу T3.
11. Обчислюється час, затрачений на сортування всіх файлів: T3 – T0.

Сортування, при якому кількість потоків дорівнює кількості вузлів обчислення. Алгоритм сортування N файлів, при якому кількість потоків дорівнює кількості вузлів обчислення має таку схему:

1. Визначається кількість файлів для оброблення: N.
2. Визначається кількість вузлів обчислення: M.
3. Створюється M потоків.
4. Розподіляється N файлів рівномірно між M потоками.
5. Запам'ятовується мітку часу T0.
6. Запускається одночасно M потоків.
7. *Установлюється лічильник I на першому файлі I = 0.
8. *Запам'ятовується мітку часу T1.
9. *Сортується I-й файл.
10. *Запам'ятовується мітку часу T2.
11. *Обчислюється час, затрачений на сортування I-го файлу: T2 – T1.
12. *Збільшується лічильник I = I+1.
13. *Якщо лічильник I менше кількості файлів у потоці – виконується крок 8.
14. *Установлюється параметр, який фіксує, що сортування завершено.
15. *Викликається функцію перевірки завершеності всіма потоками сортування.
16. **Запам'ятовується мітку часу T3.
17. **Обчислюється час, затрачений на сортування всіх файлів: T3 – T0.

Сортування, при якому кількість потоків дорівнює кількості файлів. У цьому випадку алгоритм сортування N файлів, при якому кількість потоків дорівнює кількості файлів має таку схему:

1. Визначається кількість файлів для оброблення: N.
2. Створюється N потоків.
3. Розподіляється N файлів між N потоками.
4. Запам'ятовується мітку часу T0.
5. Запускається одночасно N потоків.
6. *Запам'ятовується мітку часу T1.
7. *Сортується файл.
8. *Запам'ятовується мітку часу T2.
9. *Обчислюється час, затрачений на сортування файлу: T2 – T1.
10. *Установлюється параметр, який фіксує, що сортування завершено.
11. *Викликається функцію перевірки завершеності всіма потоками сортування.
12. **Запам'ятовується мітку часу T3.
13. **Обчислюється час, затрачений на сортування всіх файлів: T3 – T0.

* – операція виконується кожним потоком окремо.

** – операція виконується, якщо всі потоки завершили сортування.

Результати дослідження

Розглянемо такі варіанти вхідних даних: масиви мають однакову довжину; масиви мають різну довжину.

Параметри вхідних даних: тип – дійсні числа; мінімальне значення – 0; максимальне значення – 1000; знаків після коми – 2.

Конфігурація персонального комп'ютера: ОС Microsoft Windows XP Professional RU 2002 Service Pack 3; Процесор Intel(R) Core(TM) 2 Duo E8200 @ 2.66ГГц; ОЗП 2ГБ; Віртуальна пам'ять 2ГБ.

У табл. 1 наведено результати сортування масивів однакової довжини. При цьому вхідними даними є 10 файлів обсягом 25000 чисел кожний.

У табл. 1 наведені час, потрачений на сортування кожного масиву, загальний час сортування та ефективність відносно сортування одним потоком – на скільки швидше відбувається сортування відносно сортування одним потоком.

Як видно з результатів сортування двома потоками та сортування кожного файлу окремим потоком значно випереджають по швидкодії сортування одним потоком, відповідно на 42,97% та 43,6%, та мають незначну різницю між собою.

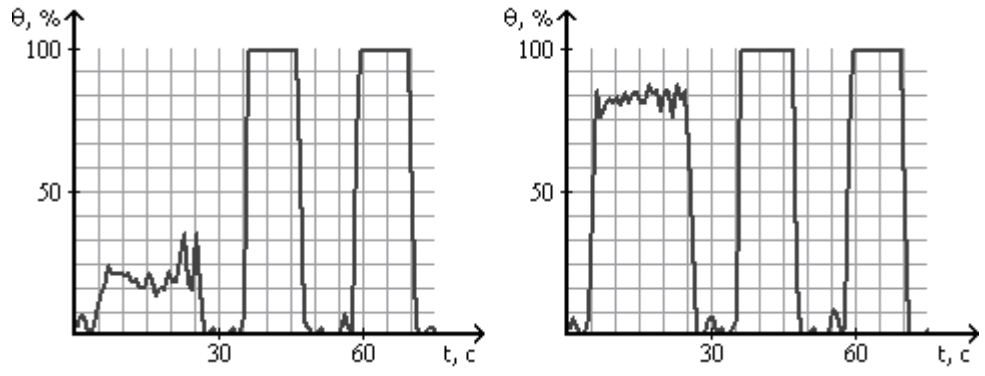
На рис. 1 показано графіки залежності навантаження θ центрального двохядерного процесора від часу t сортування масивів різної довжини. Як видно з рисунка, під час сортування одним потоком перше ядро завантажене на 25%, друге ядро – на 80%. Це показує, що перше ядро відгукується виключно на внутрішні запити системи, друге – сортує масиви. Під час сортування двома потоками та сортування кожного файлу окремим потоком обидва ядра завантажені на 100%.

У табл. 2 наведено результати сортування масивів різної довжини. При цьому вхідними даними є 10 файлів довжинами: 5000, 10000, 15000, 20000, 25000, 30000, 35000, 40000, 45000, 50000.

Таблиця 1

Результати сортування масивів однакової довжини

Файл	Сортування одним потоком (мс)	Сортування двома потоками (мс)	Сортування кожного файлу окремим потоком (мс)
input25000fdwiy.in	2500	2672	13422
input25000gdtsm.in	2500	2641	13766
input25000gkhep.in	2500	2671	13641
input25000gvjxv.in	2516	2954	13906
input25000hkeod.in	2500	2671	13297
input25000idrda.in	2484	2969	13984
input25000irpjt.in	2516	2969	13391
input25000roakx.in	2500	2687	14078
input25000sdubf.in	2515	2688	13422
input25000svkfo.in	2485	2953	13984
Загальний час	25016	14266	14109
Ефективність	—	42,97%	43,6%



а б

Рис. 1. Графіки залежності навантаження центрального процесора від часу сортування масивів однакової довжини:
а – перше ядро процесора;
б – друге ядро процесора.

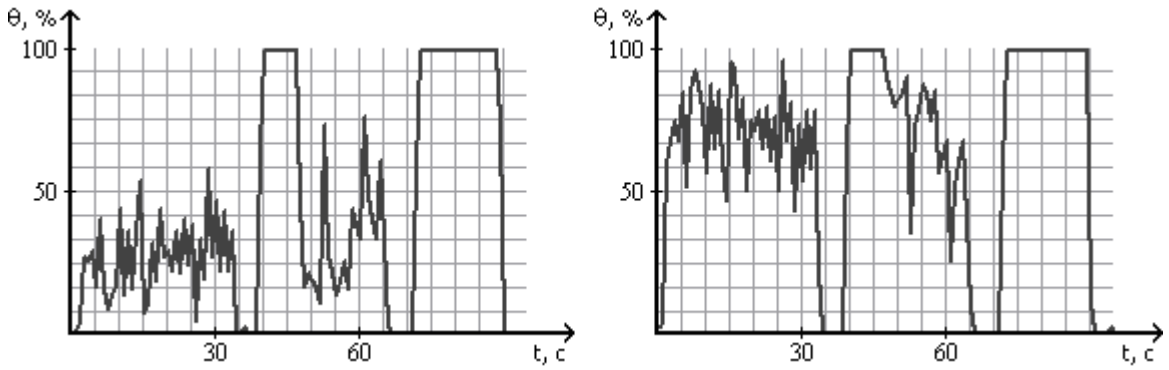
Таблиця 2

Результати сортування масивів різної довжини

Файл	Сортування одним потоком (мс)	Сортування двома потоками (мс)	Сортування кожного файлу окремим потоком (мс)
input5000.in	156	109	1032
input10000.in	469	437	2485
input15000.in	969	969	4469
input20000.in	1672	1719	7032
input25000.in	2547	2656	10313
input30000.in	3656	4219	12047
input35000.in	4875	5672	14000
input40000.in	6312	6640	16859
input45000.in	7938	8360	21078
input50000.in	9735	10250	21438
Загальний час	38329	31031	21657
Ефективність	—	19,04%	43,5%

У табл. 2 наведено результати сортування масивів різної довжини. Як видно з результатів сортування двома потоками та сортування кожного файлу окремим потоком, як і в першому дослідженні, випереджають по швидкодії сортування одним потоком, відповідно на 19,04% та 43,5%, але мають суттєву різницю між собою.

На рис. 2 показано графіки залежності навантаження θ центрального двохядерного процесора від часу t сортування масивів різної довжини.



а б

Рис. 2. Графіки залежності навантаження центрального процесора від часу сортування масивів різної довжини:
а – перше ядро процесора;
б – друге ядро процесора

Як видно з рисунка під час сортування одним потоком перше ядро завантажене на 20–50%, друге ядро – на 50–90%. Як і в першому дослідженні це показує, що перше ядро відгукується виключно на внутрішні запити системи, друге – сортує масиви. На початку сортування двома потоками обидва ядра завантажені на 100%, в кінці навантаження не рівномірно падає. Під час сортування кожного файлу окремим потоком обидва ядра завантажені на 100%.

Зазначимо, що під час першого дослідження навантаження обох ядер відбувається рівномірно, порівняно з другим дослідженням. Це можна пояснити тим, що масиви мають однакову довжину, а отже для їх сортування потрібно виділяти однакову кількість ресурсів. Під час другого дослідження навантаження розподіляється нерівномірно, що пояснюється різною довжиною масивів, що в свою чергу потребує різну кількість ресурсів.

Висновки

1. Розглянуто підхід до розроблення багатопоточних програм шляхом розпаралелювання послідовних алгоритмів та розбиття послідовної програми на підзадачі.

2. На прикладі задачі сортування масивів було продемонстровано різницю в швидкодії між послідовною та двома паралельними програмами. Дослі-

дження показали, що використання розпаралелювання, при якому кількість потоків дорівнює кількості файлів обробки є найдоцільнішим як з точки зору проектування та реалізації програми, так і за швидкодією.

3. До перспектив використання розпаралелювання, при якому кількість потоків дорівнює кількості об'єктів дослідження належить інформаційно–екстремальна інтелектуальна технологія, яка ґрунтується на максимізації інформаційної спроможності системи розпізнавання.

Література

1. The Landscape of Parallel Computing Research: A View from Berkeley / Krste Asanovic and all. – University of California, Berkeley. Technical Report No. UCB/EECS-2006-183, 2006. – 56 p.

2. Воеводин В.В. Математические основы параллельных вычислений / В.В. Воеводин. – М.: МГУ, 1991. – 345 с.

3. Воеводин В.В. Параллельные вычисления / В.В. Воеводин, Вл.В. Воеводин. – СПб.: БХВ – Петербург, 2002. – 540 с.

4. Хокни Р. Параллельные ЭВМ: Архитектура, программирование и алгоритмы / Р. Хокни, К. Джесхоуп. – М.: Радио и связь, 1986. – 392 с.

5. Patterson David A. Computer Organization and Design (Second Edition) / David A. Patterson, John L. Hennessy. – Morgan Kaufmann Publishers, 1998. – 715 p.

6. Богачев К.Ю. Основы параллельного программирования / К.Ю. Богачев. – М.: БИНОМ. Лаборатория знаний, 2003. – 342 с.

7. Подосельник А.С. Основы многопоточного, параллельного и распределенного программирования: пер. с англ. / А.С. Подосельник, Г.И. Сингаевская.– М.: Изд. дом «ВИЛЬЯМС. – 2003. – 512 с.

Поступила в редакцию 1.03.2011

Рецензент: д-р техн. наук, проф., зав. кафедри інформатики О.Ю. Соколов, Національний аерокосмічний університет ім. М.Є. Жуковського «ХАІ», Харків, Україна.

ОСОБЕННОСТИ ПРОЕКТИРОВАНИЯ МНОГОПОТОЧНЫХ ПРОГРАММ

А.С. Довбыш, А.Б. Барило

Рассмотрен подход к разработке многопоточных программ путем распараллеливания последовательных алгоритмов и разбиения последовательной программы на подзадачи в зависимости от количества узлов вычислений и объектов исследования. Проанализированы случаи распараллеливания последовательных алгоритмов, при которых количество потоков равно количеству узлов исчисления и количества объектов исследования. На примере задачи сортировки массивов показано преимущество по оперативности реализации параллельных алгоритмов исчислений по сравнению с последовательным. При этом определено, что использование распараллеливания, при котором количество потоков равно количеству файлов обработки является наиболее целесообразным как с точки зрения проектирования и реализации программы, так и по быстродействию.

Ключевые слова: программное обеспечении, алгоритм, подзадача, поток, узел вычисления, параллельная программа, последовательная программа, эффективность.

DESIGN FEATURES MULTITHREADED PROGRAMS

A.S. Dovbysh, A.B. Barylo

Article describes the multithreaded programs by parallelizing sequence algorithms and partitioning a sequential program into subtasks based on the number of nodes and computing objects of study. The cases of parallel algorithms for sequential were analyzed. The number of threads and the number of calculation nodes and the number of objects are equal. As the example of sorting arrays was shown an advantage of the implementation of parallel algorithms compared with a serial. We come to the conclusion that the use of parallelism, in which the number of threads and the number of files are equal, is the most expedient in terms of design and implementation of the program and speed.

Keywords: software, algorithm, subproblem, stream, node computing, object of study, multithreaded program, sequential program, effectiveness.

Довбиш Анатолій Степанович – д-р техн. наук, професор, зав. кафедри комп'ютерних наук, Сумський державний університет, e-mail: kras@id.sumdu.edu.ua.

Барило Олександр Борисович – аспірант кафедри комп'ютерних наук, Сумський державний університет, e-mail: alex.barylo@gmail.com.