УДК 681.5.

Іваненко П.А.[1], Дорошенко А.Ю.[2]
[1]ИПС НАНУ
[2]Київський національний університет імені Тараса Шевченка

ЗАСОБИ СТВОРЕННЯ СИСТЕМ АВТОМАТИЧНОГО НАСТРОЮВАННЯ ДЛЯ ЕФЕКТИВНОГО ВИКОНАННЯ ПРИКЛАДНИХ ПАРАЛЕЛЬНИХ ПРОГРАМ

У роботі пропонуються програмні засоби для автоматичного створення автотюнерів – програмних застосунків для оптимізації виконання прикладних паралельних програм у цільовому обчислювальному середовищі.

Ключові слова: автотюнінг, паралельні алгоритми, фреймворк, техніка переписувальних правил, прагми

Вступ

У процесі розробки будь-якого програмного забезпечення етап оптимізації є однаково важливим і складним. Особливо складним і ресурсоємним завданням є створення паралельних застосунків, які мали б бути однаково ефективними для різних мультипроцесорних платформ. Сучасним підходом до вирішення цієї проблеми є автотюнінг [1]. Автотюнер – це незалежний застосунок чи бібліотека, що породжує еквівалентні за отриманими результатами обчислень варіації оптимізованого коду застосунку й емпірично вибирає найефективніший з них за певним критерієм, найчастіше – за часом виконання [2]. Зазвичай ця оптимізація виконується в цільовому обчислювальному середовищі (ОС) один раз, а отримана варіація оптимізованого застосунка залишається найефективнішою, доки конфігурація цього середовища буде незмінною. Цей підхід дозволяє абстрагуватися від специфічних характеристик ОС на етапі розробки що у свою чергу дозволяє розробляти ефективні в будь-якому ОС застосунки. Предметом цієї роботи є побудова програмного засобу, що дозволяє майже повністю автоматизувати генерацію автотюнерів для досить широкого класу задач. Результати застосування розробленого засобу демонструються на прикладі задачі оптимізації гібридного алгоритму сортування.

Автотюнінг та його застосування

На практиці використання автотюнінга є виправданим для застосунків, ефективність обчислень яких визначається скінченною множиною параметрів (конфігурацією), значення яких незмінні для різних вхідних даних. Слід зауважити, що параметрами можуть бути й різні алгоритмічні варіації підпрограм, структур даних чи алгоритмів обходу цих структур. Завдання розробника полягає у визначенні цих параметрів, а також області їх допустимих значень.

Очевидно, що описаний клас задач є дуже широким і включає в себе застосунки для різних типів платформ: від мобільних пристроїв до кластерів. Проте найбільш широкого застосування автотюнінг набув у сфері паралельних обчислень, де завдання профілювання й оптимізації застосунка найбільш ресурсозатратне.

Найчастіше автотюнери класифікують наступним чином [6]:

– автотюнери як бібліотеки – пошук оптимального варіанту бібліотечних функцій виконується під час її інсталяції в ОС. Далі ця бібліотека використовується іншими застосунками. Відомими прикладами є бібліотеки автоматично оптимізованих підпрограм лінійної алгебри ATLAS [7] й обчислень дискретних перетворень Фур'є FFTW [8];

– автотюнери як окремі застосунки – автотюнер відокремлений від оптимізованого застосунка. На відміну від попереднього варіанту він виконує оптимізацію усього застосунка, а не деякої множини базових функцій. Саме така система програмних засобів розглядається в цій роботі. Серед аналогічних проектів можна зазначити Atune-IL [1] , що являє собою мовне розширення для автотюнінгу;
– автотюнер як частина операційної системи – такий автотюнер інтегрований в ОС і автоматично виконує оптимізацію всіх виконуваних додатків. Це рішення дуже складне з архітектурної точки зору, проте обіцяє значні переваги, такі як однократна розробка автотюнера для всього ОС, а також вирішення завдання глобальної оптимізації ОС з множиною одночасно виконуваних застосунків.

У попередній роботі авторів [3] розглядалося застосування техніки автотюнінга до задачі метеорологічного прогнозування. У результаті були досягнуті гарні показники швидкодії, проте створений автотюнер міг працювати лише з одним застосунком. У цій роботі на основі попереднього досвіду запропоновано створити універсальний програмний засіб для автоматичної генерації автотюнерів. Перейдемо до його розгляду.

TuningGenie – програмний засіб для генерації автотюнерів

TuningGenie працює з вихідним кодом застосунка й використовує метадані для породження його варіацій. Ці метадані є фактично «експертним» знанням розробника для подальшої автоматичної оптимізації й представлені прагмами (коментарями). Використання прагм дозволяє не виконувати складний аналіз залежності між даними у вихідному коді застосунку а також суттєво звужує область пошуку оптимальної конфігурації.

Загальна схема роботи автотюнера є наступною:

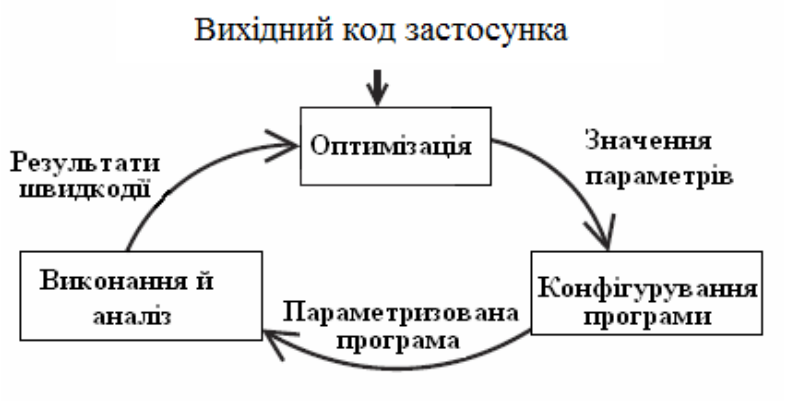

Рис. 1. Схема роботи автотюнера.

Спочатку інформація з усіх прагм у вихідному коді збирається парсером і на її основі генеруються усі можливі конфігурації застосунку. Далі для кожної конфігурації генерується відповідна варіація програми й виконуються заміри швидкодії. На основі отриманих результатів знаходиться оптимальна конфігурація й по ній генерується оптимальний варіант застосунка. Оскільки прагми є коментарями – то їх присутність не впливає на компілювання вихідного коду й ніяк не заважає розробці.

Для трансформацій вихідного коду використовується TermWare [4],[5] – потужна система, заснована на техніці переписувальних правил, яка дозволяє подавати код застосунків у вигляді термів. Використання цієї системи робить можливим генерування проміжних варіантів застосунка із структурними змінами в алгоритмах. Ця система не залежить від мови, на якій написано застосунок, але всі приклади в цій статті написані на Java.

TuningGenie підтримує на даний момент три типи прагм:

1. *tuneAbleParam* визначає межі значень числової змінної. Наприклад наступним чином для змінної `numThreads` задаються межі [1..10] з кроком 2:

```
//tuneAbleParam name=numSubTasks start=1 stop=10 step=2
int numSubTasks = 1;
```

Цей тип прагм дозволяє, наприклад, автоматично підібрати найкращу декомпозицію вхідних даних на під задачі, як це робилося у [3].

2. *calculatedValue* ініціалізує змінну значенням, яке необхідно обчислити. Завдяки цьому типу прагм можна легко додати до застосунка інформацію про ОС. Це дуже зручно, коли, наприклад, алгоритм спирається на швидкість доступу до різних типів пам'яті чи швидкість виконання базових арифметичних операцій. Усі значення таких змінних обчислюються в окремій фазі до етапу виконання оптимізації застосунка й зберігаються у базі знань TermWare.

Приклад застосування:

```
//calculatedValue name=hdReadSpeed method=
//"org.tuning.EnvironmUtils.getHDReadSpeed()"
int hdKbPerSec= 1;
```

3. *bidirectionalCycle* вказує на те, що напрям обходу циклу не має значення й може бути зміненій на обернений. Наприклад, для наступного циклу будуть випробовувані його інкрементальна й декременальна версії:

```
int[] data = new int[SIZE];
//bidirectionalCycle
for (int i=0; i <SIZE; i++) {
       doSomethingWith(data[i]);
}
```

Ця прагма надає можливість експериментування з напрямками обходу даних, що для задачі з [3] дало приріст швидкодії на майже 15%, що пояснюється різною ефективність використання процесорного кешу при зміні напрямку обходу даних.

Демонстраційний приклад

Розглянемо модифікований алгоритм сортування Quicksort який для сортування малих підзадач буде використовувати сортування прямим включенням яке, як відомо, ефективне на малих масивах:

```
void enhancedQuick(int[] a, int lowerBound, int upperBound){
   Stack stack = new Stack();
   //tuneAbleParam name=threshold start=1 stop=3000 step=5
   int threshold = 1;

   addPartitionOrSort(a, lowerBound, upperBound, stack, threshold);
        while (!stack.empty()) {
        ..........................
        }
}

void addPartitionOrSort(int[] array, int lb, int up, Stack toSort,
     int threshold) {
              if (ub - lb >= threshold) {
                    toSort.push(lb);
                    toSort.push(ub);
              } else {
                   insertionSort(array, lb, ub);
             }
    }
```

Єдиною модифікацію вихідного коду для використання TuningGenie є прагма *//tuneAbleParam name=threshold*. Далі тюнер автоматично виконує пошук і знаходить оптимальну для тестового середовища величину *threshold* = 89, з якою гібридна модифікація виявляється швидшою за класичний QuickSort приблизно на 30%.

Графік залежності часових затрат у мілісекундах на сортування масиву з 2 000 000 елементів від величини *threshold*:

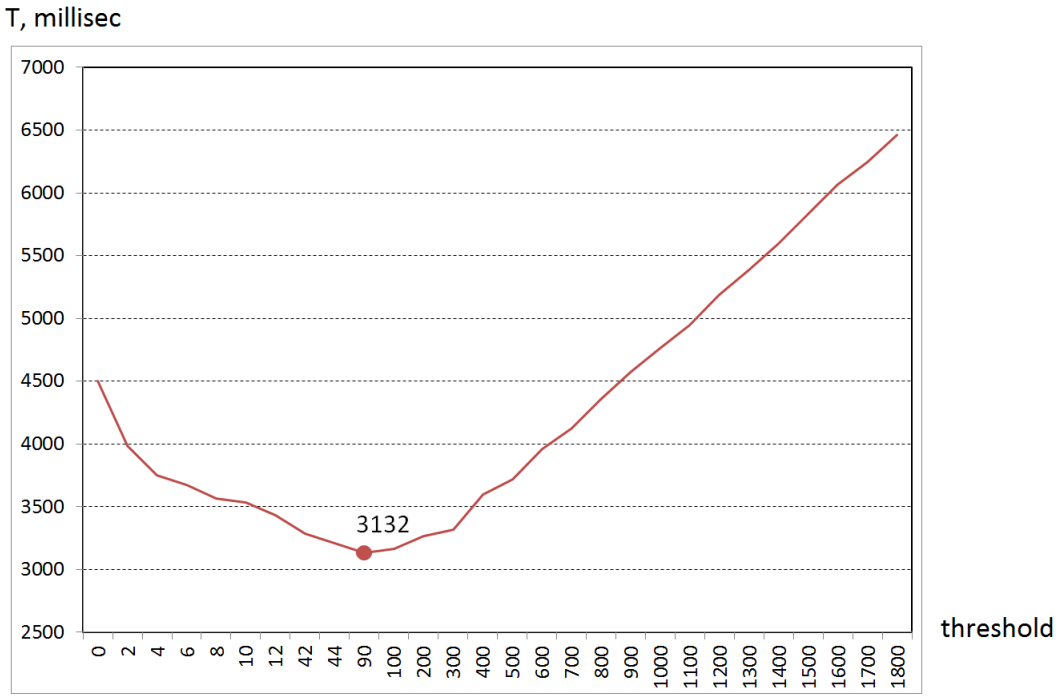

Рис. 2. Результати експерименту.

Конфігурація експериментального середовища:

– Intel® Core™ i5-2410M Processor (3M Cache, up to 2.90 GHz)
– 4 GB DDR2 RAM

Порівняння з існуючими рішеннями

Найбільш близькою за функціональністю до TuningGenie системою є Atune-IL [1]. Обидві системи можна використовувати для оптимізації застосунків написаних на будь-якій мові програмування і є незалежними від предметної області застосунка. Обидві системи містять засоби для вимірювання швидкодії застосунка й використовують прагми для передачі експертних даних від розробника. Проте для перетворення вихідного коду Atune-IL використовує StringTemplate [9],[10] який є менш гнучким засобом порівняно з TermWare. Представлення вихідного коду у вигляді термів й використання переписувальних правил для його модифікації робить можливими структурні зміни вихідного коду (наприклад, наведена раніше прагма *bidirectionalCycle*). Варто також згадати про зручність використання бази знань, що є частиною TermWare, для передачі емпірично отриманих чисельних характеристик ОС налаштовуваному застосунку (прагма *calculatedValue*).

Застосування TuningGenie в освіті

Продемонстрована в роботі програмна система автотюнінгу дозволяє просто й наочно продемонструвати емпіричний підхід до оптимізації паралельних алгоритмів. Розгляд TuningGenie включено до практичної частини курсу лекцій «Паралельні обчислювальні системи» [11]. У широкому контексті застосування TuningGenie, як і самої парадигми автотюнінгу, не обмежене розробкою паралельних програмних систем, тому ця система може бути використана в будь-якому практичному курсі для дослідження ефективності алгоритмів залежно від їх внутрішніх параметрів і властивостей обчислювального середовища.

Висновки

Автотюнінг є потужним засобом для оптимізації швидкодії паралельних застосунків, який також значно економить час їх розробки. Створення автотюнерів за рахунок використання метаданих й техніки переписувальних правил суттєво посилює методологію автоматичної оптимізації за стосунків. Розглянута у роботі система дозволяє абстрагувати розробку застосунка від специфіки ОС у якому він буде виконуватися й при цьому гарантувати оптимальність його виконання. TuningGenie дозволяє легко створювати

застосунки, що спираються на емпірично отримані дані про ОС. Також слід виокремити застосування TuningGenie для дослідження «поведінки» застосунка. Оскільки під час оптимізації зберігаються заміри швидкодії для усіх конфігурацій – це дозволяє якісно аналізувати її залежність від визначених параметрів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Schaefer C.A., Pankratius V., and Tichy W. F. Atune-IL: An instrumentation language for auto-tuning parallel applications // Euro-Par '09 Proc. 15th Int.Euro-Par Conf. on Parallel Processing Springer-Verlag Berlin, Heidelberg 2009.
2. K. Asanovic et al. The Landscape of Parallel Computing Research: A View From Berkeley, Technical Report, University of California, Berkeley, 2006.
3. Іваненко П.А., Дорошенко А.Ю. Автоматична оптимізація виконання для задачі метеорологічного прогнозування // Проблеми програмування. – 2012. – № 2-3. – С. 426–434.
4. TermWare http://www.gradsoft.ua/products/termware_rus.html
5. Anatoliy Doroshenko, Ruslan Shevchenko. A Rewriting Framework for Rule-Based Programming Dynamic Applications - Fundamenta Informaticae - SPECIAL ISSUE ON CONCURRENCY SPECIFICATION AND PROGRAMMING (CS&P 2005) Ruciane-Nide, Poland, 28-30 September 2005 Pages 95-108
6. Thomas Karcher, Christoph Schaefer, Victor Pankratius. Auto-Tuning Support for Manycore Applications - Perspectives for Operating Systems and Compilers, Technical Report, University of Karlsruhe, Karlsruhe, Germany, 2009
7. R. Whaley, A. Petitet, and J. J. Dongarra, “Automated em-pirical optimizations of software and the ATLAS project”. Parallel Computing, 27(1-2), pp. 3-35, Jan. 2001
8. M. Frigo and S. Johnson, “FFTW: An adaptive software architecture for the FFT,” Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE Interna-tional Conference on, vol. 3, pp. 1381–1384 vol.3, 1998.
9. T. Parr. The StringTemplate Homepage. http://www.stringtemplate.org/. Last accessed September 2008.
10. M. Puschel, J. Moura, J. Johnson, D. Padua, M. Veloso, B. Singer, J. Xiong, F. Franchetti, A. Gacic, Y. Voronenko, K. Chen, R. Johnson, and N. Rizzolo, “SPIRAL: Code gen-eration for dsp transforms”, Proceedings of the IEEE, vol. 93, no. 2, pp. 232–275, 2005.
11. А.Ю. Дорошенко, Курс лекцій "Паралельні обчислювальні системи"// Видавничий дім КМА. – 2003 р. – 42 с.

Стаття надійшла до редакції 20.01.2013.

Ivanenko P.[1], Doroshenko A.[2]
[1]Institute of Software Systems of NASU
[2]National Taras Shevchenko University of Kyiv

TUNINGENIE - AN AUTOTNUNING FRAMEWORK FOR OPTIMIZATION OF PARALLEL APPLICATIONS

There are proposed software tools for automatic generating autotuners – special kind of applications to optimize running parallel application software in target computing environment.

Keywords: autotuning, parallel application, framework, rule-based rewriting framework, pragma

Иваненко П.А.[1], Дорошенко А.Е.[2]
[1]ИПС НАНУ
[2]Киевский національній університет шимени Тараса Шевченка

СРЕДСТВА СОЗДАНИЯ СИСТЕМ АВТОМАТИЧЕСКОЙ ОПТИМИЗАЦИИ ДЛЯ ЭФФЕКТИВНОГО ВЫПОЛНЕНИЯ ПРИКЛАДНЫХ ПАРАЛЕЛЬНЫХ ПРОГРАМ

В роботе представлены программные средства для автоматической генерации автотюнеров – программного обеспечения для оптимизации выполнения прикладных параллельных программ целевой вычислительной среде.

Ключевые слова: автотюнинг, параллельные алгоритмы, фреймворк, техника переписывающих правил, прагмы