Машинно-залежна оптимізація в компіляторах

А.О. Іщенко, *студент,*
В.А. Бісюк, *викладач*
Кіровоградський національний технічний університет

В даний час від компілятора потрібне отримання ефективного і високонадійного коду. З точки зору розробника програма повинна легко читатися і модифікуватися. Для реалізації цієї вимоги у великих проектах необхідна модульність, основна функціональність виноситься в окремі утиліти, використовується об'єктно-орієнтоване програмування і т.д. Завдяки цьому оптимізація програмного коду - складна і комплексна задача. Додатковий рівень складності виникає через велику кількість підтримуваних архітектур і різних розширень.

В переважній більшості випадків генерація коду виконується компілятором не для всієї вихідної програми в цілому, а послідовно для окремих її конструкцій. Отримані для кожної синтаксичної конструкції вхідної програми фрагменти результуючого коду також послідовно об'єднуються в загальний текст результуючої програми. При цьому зв'язки між різними фрагментами результуючого коду в повній мірі не враховуються.

Побудований таким чином код результуючої програми може містити зайві команди і дані. Це знижує ефективність виконання результуючої програми. В принципі, компілятор може завершити на цьому генерацію коду, оскільки результуюча програма побудована, і вона є еквівалентною за змістом (семантиці) програмі на вхідній мові. Проте ефективність результуючої програми важлива для її розробника, тому більшість сучасних компіляторів виконують ще один етап компіляції - оптимізацію результуючої програми, щоб підвищити її ефективність наскільки це можливо.

В залежності від рівня представлення програми розрізняють такі види оптимізації:

– Оптимізацію на рівні вихідної мови. При цьому в результаті трансформації виходить програма, записана в тому ж самому мовою.

– Машинно-незалежну оптимізацію. В цьому випадку перетворенню піддається програма на рівні машинно-незалежного проміжного представлення, спільного для групи вхідних або машинних мов.

– Машинно-залежну оптимізацію, тобто оптимізацію на рівні машинної мови.

З точки зору ефективності найбільш кращою є машинно-залежна оптимізація, оскільки саме з її допомогою можна врахувати особливості конкретного обчислювального середовища. Нарешті, машинно-незалежна оптимізація на рівні проміжного подання є компромісом між цими двома крайніми випадками.

Оптимізація програми – це обробка, пов'язана з переупорядковуванням і зміною операцій в компілюємій програмі з метою отримання більш ефективної результуючої об'єктної програми. Оптимізація виконується на етапах підготовки до генерації і безпосередньо при генерації об'єктного та машинного коду.

В якості показників ефективності результуючої програми можна використовувати два критерії: обсяг пам'яті, необхідний для виконання результуючої програми, і швидкість виконання програми. Далеко не завжди вдається виконати оптимізацію так, щоб задовольнити обом цим критеріям. Найчастіше скорочення необхідного програмі обсягу даних веде до зменшення її швидкодії і навпаки. Тому для

оптимізації зазвичай вибирається або один із згаданих критеріїв, або якийсь комплексний критерій, заснований на них.

Машинно-залежні методи оптимізації орієнтовані на конкретну архітектуру цільової обчислювальної системи, на якій буде виконуватися результуюча програма. Як правило, кожен компілятор орієнтований на одну певну архітектуру цільової обчислювальної системи. Іноді можливо в налаштуваннях компілятора явно вказати одну з допустимих цільових архітектур. У будь-якому випадку результуюча програма завжди породжується для чітко заданої цільової архітектури.

Архітектура обчислювальної системи складається з апаратної і програмної частини та взаємозв'язків між ними з погляду системи як єдиного цілого. Поняття «архітектура» включає в себе всі особливості і апаратних, і програмних засобів.

Далі будуть розглянуті лише два основних аспекти машинно-залежної оптимізації: розподіл регістрів процесора і породження коду для паралельних обчислень.

Процесори, на базі яких будуються сучасні обчислювальні системи, мають, як правило, декілька програмно-доступних регістрів. Частина з них може бути призначена для виконання яких-небудь певних цілей (наприклад, регістр-покажчик стека або регістр-лічильник команд), інші можуть бути використані практично довільним чином при виконанні різних операцій (так звані «регістри загального призначення»). Використання регістрів загального призначення для зберігання значень операндів і результатів обчислень дозволяє домогтися збільшення швидкодії програми, так як дії над регістрами процесора завжди виконуються швидше, ніж над комірками пам'яті.

Крім загального розподілу регістрів, можуть використовуватися алгоритми розподілу регістрів спеціального характеру. Наприклад, в багатьох процесорах є регістр-акумулятор, який орієнтований на виконання різних арифметичних операцій (операції з ним виконуються або швидше, або мають більш короткий запис). Тому в нього прагнуть завантажити операнди, які найчастіше всього використовуються; він також використовується, як правило, при передачі результатів функцій та окремих операторів. Можуть бути також регістри, орієнтовані на зберігання лічильників циклів, базових покажчиків і т. п. Тоді компілятор повинен прагнути розподілити їх саме для тих цілей, на виконання яких вони орієнтовані.

Більшість сучасних процесорів допускають можливість паралельного виконання декількох операцій. Тоді компілятор може породжувати об'єктний код таким чином, щоб в ньому містилося максимально можливу кількість сусідніх операцій, всі операнди яких не залежать один від одного. Рішення такого завдання в глобальному обсязі для всієї програми в цілому не представляється можливим, але для конкретного оператора воно, як правило, полягає в порядку виконання операцій. В цьому випадку знаходження оптимального варіанта зводиться до виконання перестановки операцій (якщо вона можлива, звичайно). Причому оптимальний варіант залежить як від характеру операції, так і від кількості наявних в процесорі конвеєрів для виконання паралельних обчислень.

Таким чином використання машинно-залежної оптимізації дозволяє компілятору генерувати ефективний код з мов високого рівня для архітектур, що підтримуються, який майже не поступається по якості програмам на мові асемблера.

Список літератури

1. http://www.intuit.ru/department/sa/compilersdev/11/1.html
2. http://www.intuit.ru/department/se/inintopt/1/
3. А.Ю. Молчанов Системное программное обеспечение: Учебник для вузов – СПб.: Питер, 2003,-396.