УДК 004.415.2.043

САЛАПАТОВ В.І.

МЕТОДИ ОПТИМІЗАЦІЇ ПРОГРАМ НА РІВНІ МАШИННИХ КОДІВ

В статті наводяться методи оптимізації машинного коду на рівні мови Асемблер, які враховують систему команд та архітектурні особливості цільових процесорів. Крім того, створюється база даних операндів, які існують у програмно доступних регістрах, які можуть бути використані у наступних командах. Розглядається, також, оптимальне використання стеку стосовно мінімізації звернень до нього.

Methods of the machine code optimization on Assembler language level, which are used instruction set and architectural feature of target processors. Besides, the data base operands are created. This operands are exist in program accessed registers, which are can be used on the next commands. There are considered else the optimal stack using and minimal accessed time quantity to it.

1. Вступ

Процес породження оптимальних машинних кодів програм засобами компіляції дедалі стає все ще актуальною задачею особливо для спеціалізованих пристроїв, зокрема для цифрових сигнальних процесорів. Це обумовлено, з одного боку, вимогами швидкісної обробки даних у реальному часі, а з іншого – обмеженим розміром оперативної пам'яті. Родина цифрових сигнальних процесорів ADSP-21xx має низку архітектурних особливостей, які дозволяють суміщення кількох дій в одній команді. Сучасні засоби компіляції не забезпечують у повній мірі використання усіх таких архітектурних можливостей. Неефективно, також, використовується інформація у програмно доступних регістрах, яка міститься в кожний конкретний момент під час виконання програми. На фазі генерації кодів стандартний компілятор породжує надмірні команди пересилки даних у певні регістри, тоді як ця інформація може вже знаходитись в інших регістрах. Такі команди пересилки складають левову частку надмірності машинного коду програм. На даний час оптимізація машинних кодів програм поки що здійснюється на рівні мови Асемблер, що є досить важким, довготривалим та витратним процесом. При цьому найбільш критичні фрагменти машинного коду, як правило, перероблюються на мові Асемблер, що викликає великі труднощі й займає багато часу. Тому, створення компіляторів, які забезпечують породження оптимальних машинних кодів програм з урахуванням архітектурних особливостей процесорів та існуючої інформації в їхніх програмно доступних регістрах (ПДР) в кожний конкретний момент часу виконання програми, є актуальною задачею і являє собою як теоретичний, так і з практичний інтерес. Тому метою даної статті є опис методів породження оптимального машинного коду на останній фазі роботи компіляторів з мов програмування високого рівня.

2. Особливості машинно-залежної оптимізації програм для сигнальних процесорів родини ADSP-21xx

В системах компіляції теоретично розвинені та практично застосовуються методи машинно-незалежної оптимізації, які забезпечують оптимізацію виразів, циклів, підстановки з тимчасовими змінними, тощо. Перелічені прийоми значно покращують структуру програм на абстрактному рівні, але основна надмірність машинного коду програм виникає при його породженні на останній фазі генерації роботи компілятора. Для порівняння ефективності роботи компіляторів для цифрового сигнального процесора ADSP-2188, наведемо приклад простої програми, що складається з оператора присвоєння на мові С. Розглянемо команди, що породжують компілятори з одного боку фірми Analog Device, а з іншого - створеного за описаною методикою. Текст цієї програми на мові С має наступний вигляд.

```
main()
{//4
   int E;//6
     int A = 3;
     int B = 2;
     int C = 5;
     int D = 4;
    E = (A+B) * 5 + (C-B)*D;   //12       }
```

Компілятор фірми Analog Device породжує наступні оператори Асемблера.

```
AR =  3;    // A
I6=-5;MODIFY(I6+=M4); DM(I6+=M6)=AR;
AR=PASS 2; // B
I6= -4; MODIFY(I6+=M4); DM(I6+=M6)=AR;
AR=5; // C
I6 = -3; MODIFY(I6+=M4);DM(I6 +=M6)=AR;
AR=PASS 4; // D
I6= -2; MODIFY(I6+=M4); DM(I6 +=M6)=AR;
I6 = -4; MODIFY(I6+=M4); AY1=DM(I6+=M6); // B
I6 = -3; MODIFY(I6+=M4); AX1=DM(I6+=M6); // C
AR = AX1 - AY1;
I6 = -2; MODIFY(I6 +=M4);MY1=DM(I6+=M6); // D
MR = AR * MY1(SS);
I6 =  -5; MODIFY(I6 += M4);
AX0 = DM(I6 +=M6); // A
AR = AX0+AY1; MY0 =  5;
AY0 = MR0 ;
MR0 = AY0 ;  //-----------------------------
MR = MR+AR*MY0(SS);AR=MR0 ;  //-------------
I6 = -6;MODIFY(I6 +=M4); // ---------
DM(I6 += M6) = AR; // ---------E
```

Початкові оператори пов’язані з ініціюванням змінних і тому не підлягають оптимізації. Оператори, які виділені коментарями з пунктирами, новим компілятором спрощуються наступним чином.

```
I6 = -6;
MODIFY(I6 += M6);
MR=MR+AR*MY0(SS),
DM(I6+=M6)=MR0;
```

Як бачимо, шість операторів Асемблера, що породжуються фірмовим компілятором, замінюються на три оператори. Причому останній оператор об’єднує дії множення-додавання з пересилкою. Для більш складних програм ефективність коду буде ще більш відчутною.

Як було показано вище, фірмовий компілятор мови С не враховує ані архітектурних особливостей цифрових сигнальних процесорів родини ADSP-21xx, зокрема таких, як суміщення за певною логікою кількох дій в одній команді, умовне виконання команд та деякі інші, ані вже існуючої інформації в регістрах у кожній точці програми під час її виконанні. Інформація про особливості архітектури та системи команд цільового процесора повинна надаватись компілятору у вигляді спеціальної бази даних [1] з описом семантики кожної команди разом з операндами та способами адресації до них, розміщенням результатів виконання команди, а також зміни стану процесора. Особливість оптимізації машинних кодів полягає у тому, що необхідно описати та врахувати правила можливого суміщення кількох дій в одній машинній команді, організацію циклу повтору за лічильником та умовне виконання команд. Такі правила враховуються шляхом доповнення бази даних опису особливостей архітектури та системи команд додатковими атрибутами [2]. Також треба зазначити про особливості синтаксису Асемблера цифрових

сигнальних процесорів родини ADSP-21xx, де оператори використовують природну форму запису арифметичних виразів та їх умовне виконання. Отже основною задачею, що розглядається в даній статті є створення таких компіляторів, що породжують ефективні машинні коди.

3. Методи породження ефективних машинних кодів програм

При породженні машинного коду програм сучасні компілятори використовують перетворення їх у асемблерний еквівалент. Причому внутрішнє уявлення програми перетворюється в трьохадресовий код із стандартних фрагментів, для яких створені спеціальні універсальні програмні асемблерні фрагменти. Така універсальність породжує надлишковість машинного коду програм, оскільки для реалізації цих фрагментів використовуються наперед визначені ПДР.

Оскільки більшість команд процесорів для свого виконання використовують ПДР, то в процесі генерації машинного коду програм компілятор породжує підготовчі команди пересилок R_v^i в ці регістри певних даних як операндів для інших допоміжних команд, де v – певна змінна, а i – певний регістр. Таким чином, перед програмною реалізацією кожного наступного оператора програми маємо множину інформаційних ресурсів (ІР) $\{R_v^i\}$, яку можна використовувати як операнди для команд при реалізації цього оператора. Зауважимо, що використовуватись при цьому можуть усі припустимі ПДР. Для породження допоміжних команд спочатку здійснюється пересилка даних у вільні ПДР. Поряд з невикористаними ПДР вільними будемо вважати, також ті, інформація в яких в подальшому не буде використовуватись. Для позначки про зайнятість ПДР додано спеціальний атрибут у БД опису особливостей архітектури та системи команд цільового процесора [2]. Коли всі ПДР стають зайнятими, необхідно обрати один з існуючих ПДР, зберегти його в стеку та на його місце переслати потрібний операнд. Після програмної реалізації поточного оператора збережений у стеку ІР необхідно відновити. При зайнятості всіх наявних ПДР для чергової підготовки операнду в першу чергу обирається той регістр, ІР якого в подальшому буде використовуватись для породження машинного коду після всіх попередніх ПДР з відповідними ІР. В подальшому, якщо необхідно використати ще один ПДР, то для чергового заміщення наступного ПДР застосовується така сама методика, коли попередній заміщений ПДР не приймає участі в заміщенні. Такий перегляд здійснюється на фазі синтаксичного аналізу та машинно-незалежної оптимізації при формуванні внутрішнього подання програми. При цьому фіксується порядок використання змінної чи елементу масиву у вигляді номера оператора і обирається той ресурс, який найближчим часом не знадобиться. Такий порядок використання стеку мінімізує кількість звернень до стеку, код програми та швидкість її виконання в цілому. Це відбувається через те, що з кількох ПДР $R_i(n)$ з ресурсами R_v^i доцільно обрати той регістр, який вміщує ІР, що буде звільнений з них першим. В даному випадку аргумент n вказує на останнє використання ресурсу в регістрі R_i. Таким чином, порядковий номер оператора програми досить зручно і природно використовувати як аргумент n. Тому для заміщення ПДР обирається той $R_i(n)$, де n = min. Отже, першим методом оптимізації машинного коду є ефективне використання стеку за описаним алгоритмом.

Крім того, за наявністю потрібного ІР в одному з ПДР не потрібно готувати операнди допоміжними підготовчими командами пересилок. В разі розвинутої адресації в системі команд цільового процесора один і той самий ресурс може бути присутнім з різними методами доступу, тобто видами адресації. У цьому випадку застосовується реляційна операція пошуку за зразком для визначення потрібного ІР. При наявності множини однакових ІР з різними видами адресації необхідно обрати той ІР, для якого значення заданої цільової функції [3] буде мінімальним.

Формат команди з певною семантикою являє собою множину припустимих сполучень операндів, які визначають спосіб адресації до них. Формати команд описуються у БД опису системи команд і поєднують припустимі види адресацій для кожної конкретної команди мікропроцесора. У разі пошуку ІР для формування операнду в команді Асемблера використовується реляційна операція вибірки, яка відповідає, з одного боку, команді із заданою

семантикою, а з іншого – наявності потрібного операнду у ПДР. Другий метод оптимізації машинного коду передбачає багаторазове використання ІР, які формуються в процесі в ПДР в процесі породження машинних команд в кожному місці виконання програми.

Треба зазначити, що при породженні асемблерного еквіваленту програми кожен оператор мови Асемблер включає в себе операцію або сукупність операцій над певними операндами. Стосовно операндів, то в кожному місці програми вони або вже існують, або мають бути підготовлені за допомогою команд пересилок. Тобто, якщо усі необхідні складові для породження асемблерного еквіваленту програми знаходяться у БД ІР, то для створення оператора Асемблера необхідно виконати реляційну операцію з'єднання з тих відношень БД ІР, які відповідають обраному критерію оптимальності згідно із значенням цільової функції. Задача зводиться до пошуку в БД оптимальних ІР та наступного з'єднання їх разом із потрібною семантикою в оператор Асемблера. При цьому семантики операторів внутрішнього подання програми мають охоплювати семантики машинних команд цільового процесора. Тоді з'єднання такого головного оператора Асемблера можна представити як S $\{R_v^i\}$. За відсутністю необхідних ІР формуються підготовчі оператори Асемблера, які за наявності вільного ПДР готують операнд для головного оператора. За відсутністю вільного ПДР за допомогою реляційної операції пошуку за зразком знаходиться потрібний ПДР для заміщення та наступного відновлення. Таким чином, програмна реалізація оператора мови зводиться до породження асемблерних команд для заміщення ПДР із записом у стек, команд підготовки ІР, головних команд, що реалізують оператор мови та команди відновлення ІР із стеку. Такий порядок синтезу машинного коду програм дозволяє отримати оптимальну програму за обраним критерієм оптимальності.

4. Висновки

Проблема породження оптимального машинного коду за допомогою компіляторів, з одного боку залежить від ефективного використання архітектурних особливостей цільового процесора, а з іншого боку – від максимально повного використання ІР в кожному місці програми під час її виконання.

За даною методикою на факультеті обчислювальної техніки, інтелектуальних та управляючих систем в Черкаському національному університеті імені Богдана Хмельницького створено кросс-компілятор мови С для цифрового сигнального процесора родини ADSP-21xx, який на даний час знаходиться на дослідній перевірці і вже підтвердив справедливість обраної методики породження оптимального машинного коду програм. Машинні коді, які сьворюються через проміжну мову Асемблер, відповідають обраному критерію ефективності щодо мінімального машинного коду при максимальній швидкості виконання програм.

Список літератури

1. Салапатов В.И. Структура данных для описания семантики и синтаксиса команд целевой ЭВМ. Вісник національного технічного університету України «КПІ», Інформатика, управління та обчислювальна техніка, № 41. Київ, 2004, с. 191-198.
2. Салапатов В.І. Особливості підвищення якості програм у сигнальних процесорів. Вісник національного технічного університету України «КПІ», Інформатика, управління та обчислювальна техніка, №51. Київ, 2010. с.74-77.
3. Салапатов В.І. Структура системи синтезу оптимальних машинних кодів програм. Вісник Черкаського інженерно-технологічного університету. № 4. Черкаси. 2004. с. 128-132.