ТЕХНІЧНІ НАУКИ

Воєводін Є.В.
аспірант,
Науковий керівник: **Авраменко В.С.**
кандидат фізико-математичних наук, доцент,
Черкаський національний університет імені Богдана Хмельницького

МЕТОДИ МАШИННО-ЗАЛЕЖНИХ ОПТИМІЗАЦІЙ КОДУ АСЕМБЛЕРА

Серед практичних завдань програмної інженерії особливе місце займає задача оптимізації роботи програмного забезпечення (ПЗ). При цьому під оптимізацією розуміється модифікація ПЗ для поліпшення його ефективності.

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. Серед практичних завдань програмної інженерії особливе місце займає задача оптимізації роботи ПЗ. При цьому під оптимізацією розуміється модифікація ПЗ для поліпшення його ефективності тому для оптимізації зазвичай вибирається або один із згаданих критеріїв, або комплексний критерій, заснований на них.

Слід також зазначити, що публікації, присвячені теоретичним аспектам оптимізації ПЗ, досить рідкісні, і, як правило, зводяться до прийомів роботи з інструментальними засобами оптимізації.

Залежно від рівня представлення ПЗ розрізняють такі види оптимізації:

1. Оптимізацію на рівні вихідної мови програмування. При цьому підході в результаті трансформації породжується програма, записана в тій же самій мові.
2. Машинно-незалежна оптимізація. В цьому випадку перетворенню піддається програма на рівні машинно-незалежного проміжного представлення, спільного для групи вхідних або машинних мов.
3. Машинно-залежна оптимізація. Оптимізація з використанням певних архітектурних особливостей цільових обчислювальних систем.

З точки зору ефективності найбільш кращою є машинно-залежна оптимізація, оскільки саме з її допомогою можна врахувати особливості конкретного обчислювального середовища. В свою чергу машинно-незалежні оптимізації є добре вивченими та є невід'ємною частиною сучасних компіляторів й інтерпретаторів, тоді коли машинно-залежні оптимізації вивчені мало [1, с. 324] і є потенційно актуальним джерелом для досліджень.

Сучасні процесори мають, як правило, декілька програмно доступних регістрів, частина з яких призначена для виконання більш конкретних цілей, наприклад, SP(ESP) – stack pointer. Інші регістри можуть бути використанні для довільних цілей, такі регістри є регістрами загального призначення. Використання регістрів загального призначення для збереження операндів і результатів обрахунків дає змогу покращити швидкість виконання програми, оскільки операції над регістрами виконуються швидше [2, с. 268], ніж над комірками пам'яті.

Крім того, в багатьох процесорах не всі операції можуть бути виконані над комірками пам'яті, а тому часто виникає потреба в попередньому завантаженні операнда в регістр.

Для прикладу розглянемо фрагмент коду на мові PASCAL:

```
var
 index: integer;
begin
 for index:=1 to 10 do
 if index mod 2 <> 0 then
 begin
 writeln(index);
 end;
end.
```

Проаналізуємо шматок коду асемблеру, що був отриманий в результаті компіляції програми [3] і включає в себе входження описаної змінної «index»:

```
1. .Model Small                 19. mov ax, index
2. .Stack 100h                  20. mov ax, bx
3. .Data                        21. push bx
4. __bufrw db 255 dup (?)       22. mov bx, 2
5. index dw ?                   23. xor dx, dx
6. .Code                        24. div bx
7. Main proc                    25. xchg ax, dx
8. mov ax, @data                26. mov bx, ax
9. mov ds, ax                   27. mov dx, 0
10. mov index, 1                28. cmp bx, dx
11. mov bx, index               29. pop bx
12. mov cx, 10                  30. je else_1
13. jmp $+3                     31. mov ax, index
14. for_b_1:                    32. call sl_itoa
15. inc bx                      33. mov ah, 2
16. push bx                     34. mov dl, 13
17. push cx                     35. int 21h
18. mov index, bx               36. ...
```

Приведений програмний код можна оптимізувати, за допомогою як машинно-залежних так і машинно-незалежних оптимізації. Почнемо з того, що змінна «index» хоч і є оголошеною, але по суті регістрів процесора буде достатньо для здійснення операцій, що оперують змінною. Щоб впевнитися в

цьому замістимо всі використання цієї змінної використанням регістрів процесора. Для початку проаналізуємо входження змінної «index» в програмний код асемблеру, такими входженнями є рядки: 5, 10 11, 18, 19, 20, 29, 30, 31.

Для рядків 10 та 11 побудуємо наступний автомат, де «i» це змінна «index», а q_i – це стани що описують значення регістрів процесора та змінної «index» (рис. 1).

Якщо припустити, що «i» не є частиною стану, то тоді $q_1 = q_2$ з чого слідує, що можна мінімізувати програмний код, перейшовши із q_1 відразу в q_3. Для цього замінимо рядки 12 та 13 на mov bx, 1 тим самим зменшимо кількість використань змінної «index».

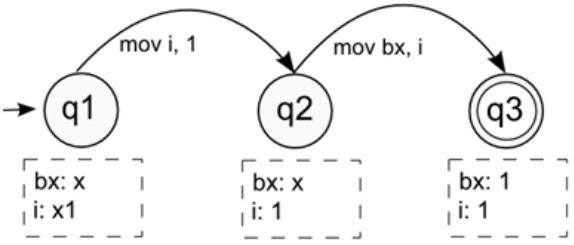

Рис. 1. Автомат станів регістрів процесора та змінних для рядків 10 та 11

Для рядків 18-20 побудуємо наступний автомат (рис. 2). Якщо не враховувати «i» як частину стану, то тоді можна або перейти зі стану q_1 в стан q_3 виконавши mov ax, bx, але тоді стан q_3 та q_4 будуть еквіваленті, що також показує надлишковість, іншим ймовірним та правильним переходом буде перехід зі стану q_1 в q_4, що дозволяє замістити використання змінної «index» на використання тільки регістрів.

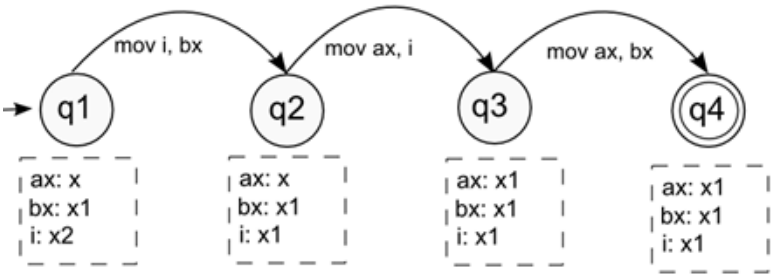

Рис. 2. Автомат станів регістрів процесора та змінних для рядків 18-20

Для рядків 18-20 побудуємо наступний автомат (рис. 2). Якщо не враховувати «i» як частину стану, то тоді можна або перейти зі стану q_1 в стан q_3 виконавши mov ax, bx, але тоді стан q_3 та q_4 будуть еквіваленті, що також показує надлишковість, іншим ймовірним та правильним переходом буде перехід зі стану q_1 в q_4, що дозволяє замістити використання змінної «index» на використання тільки регістрів.

Останньою ділянкою коду, де використовується змінна «index», є рядки 29-31. Але для того щоб перевірити чи можна оптимізувати цю ділянку коду

потрібно знати яке останнє значення було покладено в стек, рядок 21, а саме push bx показує те, що значення з регістра bx було покладено в стек. Повернувшись до попередньо описаної оптимізації можна бачити, що в регістрі bx знаходиться саме значення змінної «index» з чого слідує, що теперішнє значення в регістрі bx еквівалентне значенню змінної «index» (рис. 3).

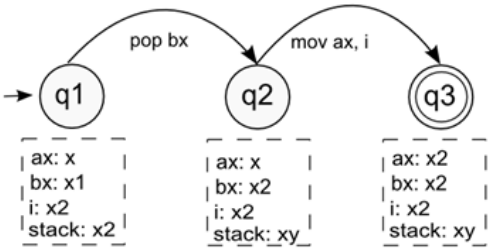

Рис. 3. Автомат станів регістрів процесора, змінних та стеку для рядків 29-31

Зі станів описаних вище видно, що для переходу з q_2 в q_3 використання змінної «index» не є обов'язковим, оскільки того ж самого ефекту можна добитись, використавши регістр bx, отже замінимо рядок 33 на mov ax, bx і позбавимось змінної «index». Оскільки всі використання змінної були замінені використанням регістрів, то можна видалити і рядок 5, що декларує змінну «index».

Описані методи перетворюють програмний код на більш ефективний за рахунок зменшення кількості операцій, які треба виконувати, а також доводять, що регістрів процесора достатньо для виконання тих же операцій, але більш ефективно.

Список використаних джерел:

1. Ахо А. В. Компиляторы: принципы, технологии, инструменты / Ахо А. В., Сети Р., Ульман Д. Д. ; – М.: Вильямс, 2001. – 768 с.

2. Молчанов А. Ю. Системное программное обеспечение / А. Ю. Молчанов. – СПб: Питер, 2010. – 400 с.

3. Машинно-зависимая и машинно-независимая оптимизация кода ассемблера [Електронний ресурс]. – Режим доступу до ресурсу: http://bibliofond.ru/view.aspx?id=455913.