

Оптимизация программ

Оптимизация программы - это изменение компилируемой программы (в основном переупорядочивание и замена операций) с целью получения более эффективной объектной программы.

Используются два критерия эффективности результирующей программы :

- **скорость** выполнения программы и
- **объем памяти**, необходимый для выполнения программы.

В общем случае задача построения оптимального кода программы алгоритмически неразрешима. К тому же, компилятор обладает весьма ограниченными средствами анализа семантики входной программы в целом.

Основная оптимизация программы должна производится программистом.

Принципиально различаются два основных вида оптимизирующих преобразований:

- **машинно-независимые преобразования** исходной программы,
- **машинно-зависимые преобразования** результирующей объектной программы.

Оптимизация может привести к изменению смысла программы. Например, в случае исключения из программы вызова функции с "побочным эффектом".

У современных компиляторов существует возможность выбора критерия оптимизации и отдельных методов оптимизации.

Машинно-независимые оптимизирующие преобразования

Машинно-независимые преобразования исходной программы производятся в основном над ее внутренним представлением и основаны на известных математических и логических преобразованиях.

1. Удаление недостижимого кода.

(задача компилятора найти и убрать его).

Пример:

```
if (1)
    S1;
else
    S2;
```

\Rightarrow

```
S1;
```

Машинно-независимые оптимизирующие преобразования

2. Оптимизация линейных участков программы.

В современных системах программирования профилировщик на основе результатов запуска программы выдаёт информацию о том, на какие её линейные участки приходится основное время выполнения.

а) Удаление бесполезных присваиваний.

$$a = b * c; d = b + c; a = d * c; \Rightarrow d = b + c; a = d * c;$$

Однако, в следующем примере эта операция уже не бесполезна:

$$p = \& a; a = b * c; d = * p + c; a = d * c;$$

б) Исключение избыточных вычислений.

$$\begin{aligned} d = d + b * c; a = d + b * c; c = d + b * c; & \Rightarrow \\ t = b * c; d = d + t; a = d + t; c = a; \end{aligned}$$

в) Свертка объектного кода (выполнение во время компиляции тех операций исходной программы, для которых значения операндов уже известны).

$$i = 2 + 1; j = 6 * i + i; \Rightarrow i = 3; j = 21;$$

Машинно-независимые оптимизирующие преобразования

г) **Перестановка операций** (для дальнейшей свертки или оптимизации вычислений).

$$a = 2 * b * 3 * c; \Rightarrow a = (2 * 3) * (b * c);$$

$$a = (b + c) + (d + e); \Rightarrow a = (b + (c + (d + e)));$$

д) **Арифметические преобразования** (на основе алгебраических и логических тождеств).

$$a = b * c + b * d; \Rightarrow a = b * (c + d);$$

$$a * 1 \Rightarrow a, \quad a * 0 \Rightarrow 0, \quad a + 0 \Rightarrow a .$$

е) **Оптимизация вычисления логических выражений.**

$$a \parallel b \parallel c \parallel d; \Rightarrow a, \text{ если } a \text{ есть true.}$$

Но! $a \parallel f(b) \parallel g(c)$ не всегда a (при $a = \text{true}$),
может быть побочный эффект.

Машинно-независимые оптимизирующие преобразования

3. Подстановка кода функции вместо ее вызова в объектный код.

Этот метод, как правило, применим к простым функциям и процедурам, вызываемым непосредственно по адресу, без применения косвенной адресации через таблицы RTTI (Run Time Type Information).

Некоторые компиляторы допускают применять метод только к функциям, содержащим последовательные вычисления без циклов.

Язык C++ позволяет явно указать (`inline`), для каких функций желательно использовать `inline`-подстановку.

Машинно-независимые оптимизирующие преобразования

4. Оптимизация циклов.

а) Вынесение инвариантных вычислений из циклов.

```
for (i = 1; i <= 10; i++)  
    a [i] = b * c * a [i];            $\Rightarrow$   
d = b * c; for (i = 1; i <= 10; i++)  
    a [i] = d * a [i];
```

б) Замена операций с индуктивными (образующими арифметическую прогрессию) переменными (как правило, умножения на сложение).

```
for (i = 1; i <= N; i++)  
    a [i] = i * 10;            $\Rightarrow$   
t = 10; i = 1; while (i <= N) {  
    a [i] = t; t = t + 10; i++;  
}
```

```
s = 10; for (i = 1; i <= N; i++) {  
    r = r + f (s); s = s + 10; }            $\Rightarrow$   
s = 10; m = N * 10; while (s <= m) {  
    r = r + f (s); s = s + 10; }
```

(избавились от одной индуктивной переменной).

Машинно-независимые оптимизирующие преобразования

в) Слияние циклов.

```
for (i = 1; i <= N; i++)  
    for (j = 1; j <= M; j++)  
        a [i] [j] = 0;    ⇒
```

```
k = N * M;  
for (i = 1; i <= k; i++)  
    a [i] = 0; (только в объектном коде!)
```

г) Развертывание циклов (можно выполнить для циклов, кратность выполнения которых известна на этапе компиляции).

```
for (i = 1; i <= 3; i++)  
    a [i] = i;    ⇒
```

```
a [1] = 1;  
a [2] = 2;  
a [3] = 3;
```

Машинно-зависимые оптимизирующие преобразования

Машинно-зависимые преобразования результирующей объектной программы зависят от архитектуры вычислительной системы, на которой будет выполняться результирующая программа. При этом может учитываться объем кэш-памяти, методы организации работы процессора

Эти преобразования, как правило, являются "ноу-хау", и именно они позволяют существенно повысить эффективность результирующего кода.

1. Распределение регистров процессора.

Использование регистров общего назначения и специальных регистров (аккумулятор, счетчик цикла, базовый указатель) для хранения значения операндов и результатов вычислений позволяет увеличить быстродействие программы.

Доступных регистров всегда ограниченное количество, поэтому перед компилятором встает вопрос их оптимального распределения и использования при выполнении вычислений.

Машинно-зависимые оптимизирующие преобразования

2. Оптимизация передачи параметров в процедуры и функции.

Обычно параметры процедур и функций передаются через стек. При этом всякий раз при вызове процедуры или функции компилятор создает объектный код для размещения ее фактических параметров в стеке, а при выходе из нее - код для освобождения соответствующей памяти.

Можно уменьшить код и время выполнения результирующей программы за счет оптимизации передачи параметров в процедуру или функцию, передавая их **через регистры** процессора.

Реализация данного оптимизирующего преобразования зависит от количества доступных регистров процессора в целевой вычислительной системе и от используемого компилятором алгоритма распределения регистров.

Недостатки метода:

- оптимизированные таким образом процедуры и функции не могут быть использованы в качестве **библиотечных**, т.к. методы передачи параметров через регистры не стандартизованы и зависят от реализации компилятора.
- этот метод не может быть использован, если где-либо в функции требуется выполнить **операции с адресами** параметров.

Языки Си и С++ позволяют явно указать (**register**), какие параметры и локальные переменные желательно разместить в регистрах.

Машинно-зависимые оптимизирующие преобразования

3. Оптимизация кода для процессоров, допускающих распараллеливание вычислений.

При возможности параллельного выполнения нескольких операций компилятор должен породить объектный код таким образом, чтобы в нем было максимально возможное количество соседних операций, все операнды которых не зависят друг от друга.

Для этого надо найти оптимальный порядок выполнения операций для каждого оператора (переставить их).

$$a + b + c + d + e + f; \Rightarrow$$

для одного потока обработки данных: $(((((a + b) + c) + d) + e) + f);$

для двух потоков обработки данных: $((a + b) + c) + ((d + e) + f);$

для трех потоков обработки данных: $(a + b) + (c + d) + (e + f);$