Московский физико-технический институт
(государственный университет)
Физтех-школа радиотехники и компьютерных технологий
Кафедра информатики и вычислительной техники

Оптимизация конструкций switch методами глобального планирования

Выпускная квалификационная работа
(бакалаврская работа)

Выполнил: Марусов А. Э., 613 гр.

Научный руководитель: к.т.н., Ермолицкий А. В.

Цель

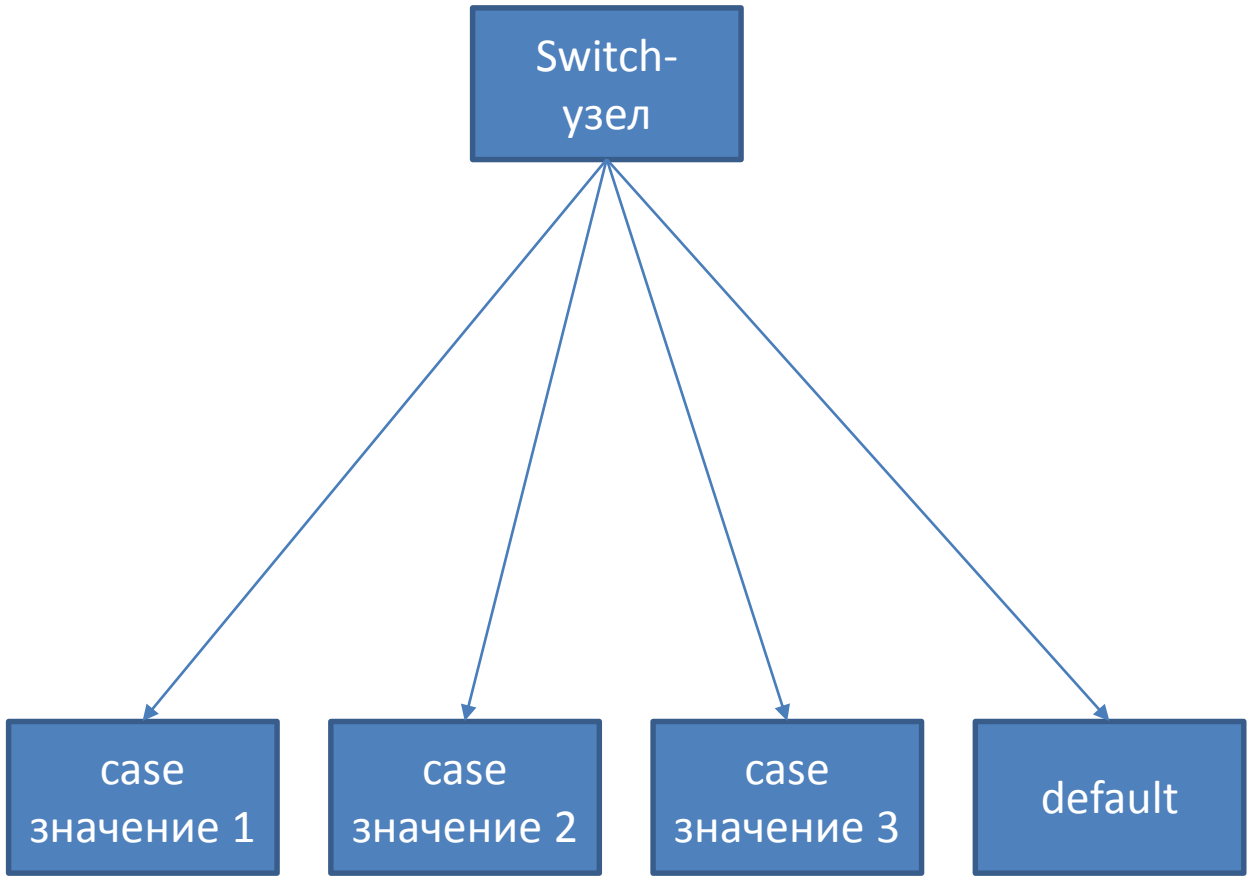
Сокращение времени исполнения конструкции switch методами глобального планирования.

Задачи

- Доработать оптимизацию gsh для применения к конструкции switch.
- Отладить доработки на задачах SPEC CPU 2017 rate.
- Произвести анализ влияния внесённых изменений на целевые программы.

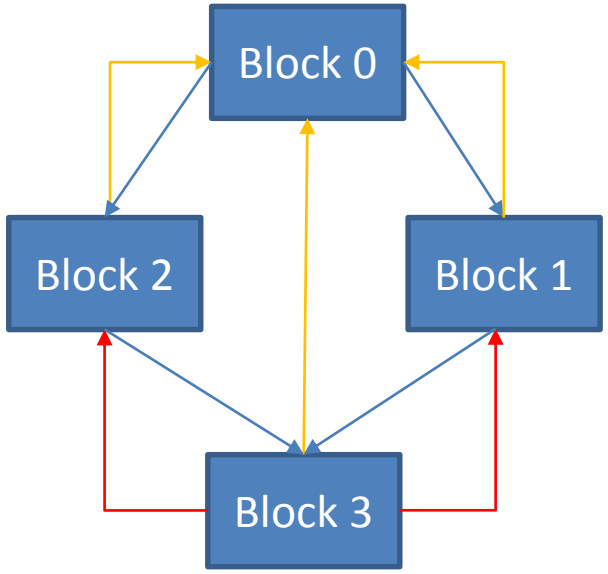
Конструкция switch в языках C\C++

```
switch (<выражение>)
{
   case значение1:
       Выполнить, если <выражение> == значение1
       break;

   case значение2:
       Выполнить, если <выражение> == значение2
       break;

   case значение3:
       Выполнить, если <выражение> == значение3
       break;
   …

   default:
     выполнить, если ни один вариант не подошел
     break;
 }
```



Представление конструкции switch в графе потока управления

Стратегии переноса в компиляторе lcc



Глобальное планирование - перемещение операций между линейными участками (изображены в виде прямоугольников). В компиляторе lcc оптимизация, реализующая глобальное планирование, называется ***gsh***.

Стратегии переноса операций в gsh:

Вынос в предшественники узла

Вынос в доминируюший узел

Критический путь(critical path) - самый длинный путь в *графе зависимостей*, т.е. в графе, отображающем зависимости между инструкциями.

Стратегия выноса в ***предшественники*** узла заключается в выносе одних и тех же операций, находящихся на критическом пути, из текущего узла во все его предшественники.

Узел d ***доминирует*** над узлом n, если любой путь от входного узла к n проходит через d. Стратегия выноса в ***доминирующий*** узел заключается в выносе операций в узел, который доминирует над текущим узлом. Предшественник узла не обязан быть доминирующим узлом и наоборот (на приведенном рисунке Block 0 является доминирующим для Block3, но не предшествующим узлом).

Глобальное планирование позволяет ускорять код за счет сокращения критического пути в узлах, из которых выносятся операции. Кроме того, вынос операций чтения позволяет уменьшить кол-во блокировок за счёт увеличения расстояния между чтением и использованием его результата.

Алгоритм переноса в компиляторе Icc

Цикл по всем узлам управляющего графа
{
    Цикл по всем стратегиям переноса
    {
        Найти пути переноса для данной стратегии

        Цикл по всем стратегиям выбора операций

        (Существующие стратегии выбора операций: выбор первой ШК, выбор операции на критическом пути)
        {
            Найти операции переноса и выполнить перенос пока есть что переносить
            {
                Планируются узлы вноса и выноса (т.е. моделируется вынос операции) и
                это планирование сравнивается с тем, что было до переноса. Если время
                исполнения программы уменьшилось, то перенос выгоден и перенос завершается.
                Иначе перенос отменяется.
            }
        }
    }
}

Доработка оптимизации gsh для применения к конструкции switch

Недостатки алгоритма переноса в компиляторе lcc

1. Операции выносятся из отдельных case-ов без анализа наличия *общих подвыражений.*

Общие подвыражения - это части выражений, которые всегда вычисляют одно и то же значение.

Пример: в уравнениях x = ((a+b)*c) и y = ((a+b)/c) общим подвыражением будет выражение a+b.

2. Существующие оптимизации для больших switch-ей (т.е. с количеством альтернатив от 10 штук) неэффективны при отсутствии ***динамического профиля***, т.е. без информации, получаемой во время исполнения программы.

Доработка оптимизации gsh для применения к конструкции switch

Нахождение операций переноса

Цикл по всем ***case***-ам узла switch
{
Обход операций в данном ***case***
Если операция является операцией чтения и находится на критическом пути,
то занести ее в список ***spisok***
}
Цикл по всем операциям ***oper*** из списка ***spisok***
{
Если операция ***oper*** помечена маркером
то начать следующую итерацию цикла

Создать список ***equiv_list***
Поместить операцию ***oper*** в ***equiv_list***

Цикл по всем операциям ***temp_oper*** списка ***spisok***, начиная с операции, следующей за ***oper***
{
если [(***temp_oper*** не помечена маркером) && (имя ***temp_oper*** совпадает с именем ***oper***) && (адрес аргумента ***temp_oper*** совпадает с ***oper***)]
то пометить операцию ***temp_oper*** маркером и поместить в список ***equiv_list***
}
Если размер ***equiv_list*** больше 2, то список ***equiv_list*** поместить в итоговый ***spisok_spiskov***
}

Отсортировать ***spisok_spiskov*** по убыванию длины подсписков

Оставить в итоговом ***spisok_spiskov*** первые 32 подсписка. Это соответствует 8 ШК по 4 команды в каждой.

Доработка оптимизации gsh для применения к конструкции switch

Вынос операций в switch-узел

Для каждой пачки чтений (LD) из списка spisok_spiskov скопировать одно чтение в switch-узел с переименованием результата, и все чтения заменить на пересылку регистра (MOV). При этом скопированное чтение необходимо поставить в полуспекулятивный режим исполнения.

{

До выноса

```
switch()
{
  case 1: LD ... -> Vs1
  case 2: LD ... -> Vs2
  case 3: LD ... -> Vs3
   ...
}
```

После выноса

```
LD.s ... -> Vs0
switch()
{
  case 1: MOV Vs0 -> Vs1
  case 2: MOV Vs0 -> Vs2
  case 3: MOV Vs0 -> Vs3
}
```

}

При создании пересылок необходимо корректировать аналитические структуры данных: Def-Use граф, граф потока данных и разметку времен раннего/позднего планирования, используя штатные функции коррекции.

Доработка оптимизации gsh для применения к конструкции switch

Особенности выноса операций в switch-узел

1. Сохранение потока данных между вынесенными операциями и их информационными преемниками достигается за счёт создания ***пересылок*** (MOV), которые обеспечивают запись из результата первого чтения в регистры результатов удаленных операций чтения.

2. Сохранение прочих потоков данных.
Вынос операции чтения может затереть результат сторонней операции, если регистры их результатов совпадут.
Пример такой ситуации показан ниже

До переноса

```
DEF -> x;
switch(...) {
   case 1: LD ... -> x; break;
   case 2: USE x; break;
}
```

После

```
DEF -> x;
LD ... -> x
switch(...) {
   case 1: break;
   case 2: USE x; break;
}
```

В этом примере операция LD затирает результат операции DEF.
Переименование результата первого чтения позволяет избежать перезатирания регистров и сохранить таким образом прочие потоки данных.

Анализ влияния внесённых изменений на целевые программы

Экспериментальные результаты

В рамках дипломной работы проведена оценка эффективности оптимизации на пакете тестов производительности SPEC CPU 2017 (rate), скомпилированных в базовом режиме. Замеры производились на вычислительной машине Эльбрус-801. Зафиксировано ускорение выполнения на двух задачах:

Задача	Ускорение
500.perlbench	1.5%
520.omnetpp	0.5%

Ускорение задачи 500.perlbench достигнуто за счёт применения оптимизации gsh к switch на 115 альтернатив из самой горячей функции задачи S_regmatch . В этом switch есть 4 горячие альтернативы, на которые приходится суммарно 66% переходов. Из всех этих альтернатив были вынесены чтения, стоящие на критическом пути. Всего было вынесено 29 пачек чтений.

В задаче 520.omnetpp удалось ускорить switch на 6 альтернатив в одном из самых горячих методов EtherMAC::printState. В каждой из альтернатив есть чтение переменной Simulation::evPtr, стоящее на критическом пути. Все эти чтения были вынесены в switch-узел.

Результаты

1. Усовершенствован алгоритм переноса операций для применения к конструкции switch.

2. Завершена отладка на задачах SPEC CPU 2017 rate и на тестах оперативного пакета тестирования. На данный момент ведётся работа над внедрением доработок в компилятор lcc.

3. Произведен анализ влияния данной оптимизации на целевые программы. Зафиксировано ускорение выполнения до 1.5% на задачах SPEC CPU 2017 rate.