Московский государственный университет имени М. В. Ломоносова

Факультет Вычислительной Математики и Кибернетики
Кафедра Системного программирования

Курсовая работа
"Разработка и реализация алгоритма подстановки выражений в последовательных программах с помощью графа потока управления"

Выполнил:
студент 3 курса 327 группы
Кочармин Михаил Дмитриевич

Научный руководитель:
доцент Бахтин Владимир Александрович

Москва, 2023

Содержание

1 Введение

Данная работа проводилась в рамках разработки программного обеспечения САПФОР – системы для автоматизации распараллеливания Фортран-программ. Целью данной работы является добавление в САПФОР реализаций алгоритмов подстановки выражений, анализа живых переменных и приватных переменных.

Система САПФОР представляет собой набор проходов – отдельных стадий выполнения, которые последовательно запускаются с целью сбора информации и/или преобразования текста входной программы. Соответсвенно, в систему были добавлены три прохода с названиями `SUBST_ANALYSIS_IR`, `LIVE_ANALYSIS_IR` и `PRIVATE_ANALYSIS_IR`.

2 Постановка задач

2.1 Подстановка выражений

Подстановка выражений заключается в замене скалярных переменных в выражениях на другие, эквивалентные им, скаляры или выражения. При этом, стоит цель максимировать количество замен в каждой цепочке подстановок.

Пример:

```
...
DO I = 1, N
  DO J = I, M
    P = J + I
    Q = J - I

    F = P / Q

    A(2*P, 3*Q) = F
  ENDDO
ENDDO
...
```

Листинг 1: До подстановки

```
...
DO I = 1, N
  DO J = I, M
    P = J + I
    Q = J - I

    F = (J + I) / (J - I)

    A(2*(J+I), 3*(J-I)) =
          (J+I)/(J-I)
  ENDDO
ENDDO
...
```

Листинг 2: После подстановки

Одной из целей данного прохода является удаление зависимостей по данным, которые могут мешать распараллеливанию циклов.

2.2 Анализ живых переменных

Класс живых переменных определяется следующим образом [2, с. 643]:

Определение 2.2.1 *Предположим, что трехадресная команда i присваивает значение переменной x. Если команда j имеет x в качестве операнда и управление может перейти от i к j по пути, на котором нет промежуточных присваиваний x, то команда j* ***использует*** *значение x, вычисленное в команде i.*

Определение 2.2.2 *При этом, переменная x называется* ***живой****, или* ***активной*** *(**live**) в команде i.*

Аналогично можно определить активность переменной не для конкретной инструкции, а для любой точки программы.

Задача прохода анализа живых переменных заключается в определении для каждого базового блока B множеств $IN_{live}[B]$ и $OUT_{live}[B]$ - множеств живых переменных соответственно перед первой и после последней инструкции рассматриваемого блока.

Для практической пользы, вместе с каждой живой переменной нужно сохранить информацию о том, в каких инструкциях используется значение переменной из данной точки. При этом, в целях экономии ресурсов, хранить номера только тех инструкций, в которых значение будет использоваться в первый раз.

Более формально, если I - множество номеров всех инструкций, в который используется значение x из точки t перед первой (или после последней) инструкции базового блока, то нужно хранить только подмножество $I_0 \subseteq I$:

$$I_0 = \{\ i \mid \forall i \in I \text{ существует путь от } t \text{ до } i, \text{ не содержащий } j\ \forall j \in I\ \}.$$

2.3 Анализ приватных переменных

Определение 2.3.1 ***Приватной*** *переменной x цикла c, называется переменная, значение которой можно локализовать в рамках одного витка цикла [4].*

Это значит, что значение переменной x после любого витка цикла не будет использоваться ни одним другим витком, оно не будет использоваться после цикла и не используется значение x, которое она имела до цикла. При этом практический интерес имеют только те приватные переменные, которые имеют в цикле хотя бы одно переопределение.

Определение 2.3.2 *Также выделяется класс* ***lastprivate-переменных*** *(или* ***приватных по выходу****) - это приватные переменные, для которых нарушается вышеописанное требование для последнего витка, то есть значение такой переменной используется после цикла.*

Задача прохода анализа приватных переменных - для каждого цикла построить множества приватных и приватных по выходу переменных.

2.4 Входные данные

В качестве входных данных для прохода подстановки выражений предоставляются:

- внутреннее представдение программы в виде графа базовых блоков (граф потока управления)
- результаты анализа достигающих определений для этого графа потока управления
- внутреннее представление программы в виде дерева операторов и выражений (часть библиотеки *S*age [3])

Результаты анализа достигающих определений поставляются в виде словарей $IN_{RD}[B]$ и $OUT_{RD}[B]$ для каждого базового блока B. Ключами данного словаря являются переменные, а значениями - множества номеров инструкций, которая порождает определимое, достижимое в точке соответственно до или после базового блока B.

В проходе анализа живых переменных используется только граф потока управления, а для прохода анализа приватных - граф потока управление, Sage-дерево [3] и результат анализа живых переменных.

3 Решение

3.1 Подстановка выражений

Алгоритм подстановки выражений работает отдельно с каждым базовым блоком. Для каждой инструкции $i = 1..m$ вычисляется словарь $available[i]$. Ключами в нём являются переменные, а значениями - выражения, которыми можно заменить данные переменные в точке сразу после этой инструкции.
Пример:

```
[1] P = I - J
[2] Q = I + J
[3] F = P / Q
[4] REF P
[5] REF Q
[6] STORE A 2 F
```

$available[1] = \{P \to "I - J"\}$
$available[2] = \{$
$\quad P \to "I - J"\ ,$
$\quad Q \to "I + J"$
$\}$
$available[3] = \{$
$\quad P \to "I - J"\ ,$
$\quad Q \to "I + J"\ ,$
$\quad F \to "P\ /\ Q"$
$\}$
$available[4, 5, 6] = available[3]$

Сначала, перед итерацией по инструкциям блока, по множеству IN_{RD} строится базовый словарь $available[0]$. Потом для каждой операции с номером $i = 1..m$ по $available[i-1]$ строится $available[i]$ следующим образом:

1. $available[i] = available[i-1]$
2. Пусть инструкция i создаёт новые определения $a_1 \to e_1, ..., a_k \to e_k$, и определяет переменные $b_1, ..., b_r$ выражениями, которые невозможно скопировать в другое место (например, если выражение сожержит вызов функции)
3. $\forall j$ если в $available[i]$ есть запись $a_j \to expr_j$ (или $b_j \to expr_j$), то во всех хранимых в $available[i]$ выражениях делается замена b_j (или a_j) на $expr_j$
4. $\forall j$ из $available[i]$ удалются все записи с ключём a_j (или b_j) или те, которые содержат a_j (или b_j) в выражении-значении
5. $\forall j$ в $available[i]$ добавляется запись $a_i \to e_i$

Таким образом, с помощью $available[i-1]$ можно выполнить подстановку переменных в i-ой инструкции.

3.2 Анализ живых переменных

Введём для базового блока два множества def_B и use_B так же, как и в [ссылка на драгонбук]:

- def_B — множество переменных, определенных (т.е. получающих значения) в блоке B до любых их использований в этом блоке;
- use_B — множество переменных, значения которых могут использоваться в блоке B до любых определений этих переменных.

Построив для каждого базового блока B множества def_B и use_B можно записать соотношения для $IN_{live}[B]$ и $OUT_{live}[B]$ [2, с. 734]:

$$IN_{live}[B] = use_B \cup (OUT_{live}[B] - def_B)$$

$$OUT_{live}[B] = \bigcup_{S - \text{ преемник } B} IN_{live}[S]$$

С граничным условием

$$OUT_{live}[\text{Выход}] = \varnothing$$

Алгоритм основан на добавлении элеметов в множества $IN_{live}[B]$ и $OUT_{live}[B]$ до тех пор, пока хотя бы одно множество $IN_{live}[B]$ будет пополнено.

3.3 Анализ приватных переменных

Имея результаты анализа живых переменных, найти для цикла L множество приватных переменных $PRI(L)$ можно построив вспомогательное множество $USE(L)$, определяемое следующим образом: переменная x входит в $USE(L)$ тогда и только тогда, когда существует инструкция, которая может использовать значение переменной x с какого-то из предыдущих витков цикла L или значение, которое имело x при входе в этот цикл.

Введём множество $PRI^+(L)$:

$$PRI^+(L) = DEF(L) - USE(L)$$

Где $DEF(L)$ - множество всех переменных, которые могут переопределяться в цикле L.

Множество $PRI^+(L)$ является объединением множеств приватных и приватных по выходу переменных.

Таким образом, множесто приватных по выходу переменных

$$PRI_{last}(L) = \{\ x \mid x \in PRI^+(L) : x \text{ жива на выходе из цикла } L\}$$

и

$$PRI(L) = PRI^+(L) - PRI_{last}(L).$$

4 Возникшие проблемы

Подстановка выражений и массивы

Поскольку достигающие определения строятся только для скаляров, при подстановке нет возможности использовать выражения, содержащие массивы, из других базовых блоков.

Циклы в графе вызова процедур

Первая проблема связана с циклами в графе вызова процедур. Для построения множества def_B и use_B нужно уже иметь результат прохода анализа живых переменных для каждой вызываемой функции. В текущей реализации обработка программ с циклами в графе вызова процедур не реализовано, поэтому предполагается, что этот граф является деревом, и анализ проходит функции в направлении от листьев к корню.

Разрастание занятой памяти

При анализе больших проектов с большим количеством глобальных переменных, для сохранения результатов прохода жиывх переменных требуется слишком много памяти. Чтобы это исправить, пришлось изменить структуру данных, пожертвовав скоростью работы этого прохода.

5 Результаты

В результате проведённой работы вышеописанные алгоритмы были успешно реализованы и интегрированы в систему САПФОР. Добавленный код был тщательно протестирован на многочисленных проектах, в том числе на пакете NAS Parallel Benchmarks[1]:

Тест	количество подстановок	количество записей о живых переменных	количество приватных переменных
BT	2486	72411	348
CG	82	4428	54
EP	33	1282	20
FT	228	3519	98
LU	3045	60864	406
MG	304	7504	71
SP	2269	117580	562

Результаты проходов на тестах из NAS Parallel Benchmarks

Список литературы

[1] *NAS Parallel Benchmarks*. URL: https://www.nas.nasa.gov/software/npb.html.

[2] Альфред В. Ахо и др. *Компиляторы: принципы, технологии и инструментарий, 2-е изд.* 2006.

[3] *Библиотека Sage++*. URL: http://www.extreme.indiana.edu/sage/.

[4] А.С. Колганов и Н.Н. Королев. "Статический анализ приватных переменных в системе автоматизированного распараллеливания Фортран-программ". В: *Параллельные вычислительные технологии (ПаВТ'2018)*. URL: http://omega.sp.susu.ru/pavt2018/short/018.pdf.