Домашнее задание №1 по курсу "Введение в тензорные компиляторы": Промежуточное представление и оптимизации над ним

Айдар Байсуваков

1 Описание программы.

В программе описана функция, которая по заданному алгоритму вычисляет число, основываясь на значениях массива целых чисел **a** и значениях целых аргументов **b** и **c**.

Говоря подробнее про алгоритм, для всех чисел **i** от **b** (включительно) до **c** (не включительно), не кратных 4, вычисляется сумма элементов массива **a** с индексами от **b** до **i** (обе границы включительно), умноженных на 30 и к которым прибавлено число 3. Все данные суммы суммируются и возвращаются из функции.

Код программы представлен ниже.

```
int function (int * a , int b , int c) {
    int result = 0;
    for ( int i = b ; i < c ; ++i ) {
        int x = 5;
        if ( i % 4 == 0)
            continue;
        for ( int j = b ; j <= i ; ++j )
            result += 2 * 3 * a[j] * x + 1 + 2;
    }
    return result;
}
```

2 Промежуточное представление Generic IR.

Для начала построим Generic IR в не SSA форме, затем аккуратно приведем его в SSA форму.

```
int function (int * a , int b , int c) {
    bb0:
        result = 0;
        i = b;
        goto bb1;
    bb1:
        if (i >= c)
            goto bb7;
        else
            goto bb2;
    bb2:
        x = 5;
        if (i % 4)
            goto bb6;
        else
            goto bb3;
    bb3:
        j = b;
```

```
        goto bb4;
    bb4:
        if (j > i)
            goto bb6;
        else
            goto bb5;
    bb5:
        t = 2 * 3;
        addr = a + j;
        val = *addr;
        t = t * val;
        t = t * x;
        result = result + t;
        result = result + 1;
        result = result + 2;
        j = j + 1;
        goto bb4;
    bb6:
        i = i + 1;
        goto bb1;
    bb7:
        return result;
}
```

3 Граф потока управления.

На основе данного Generic IR построим CFG;

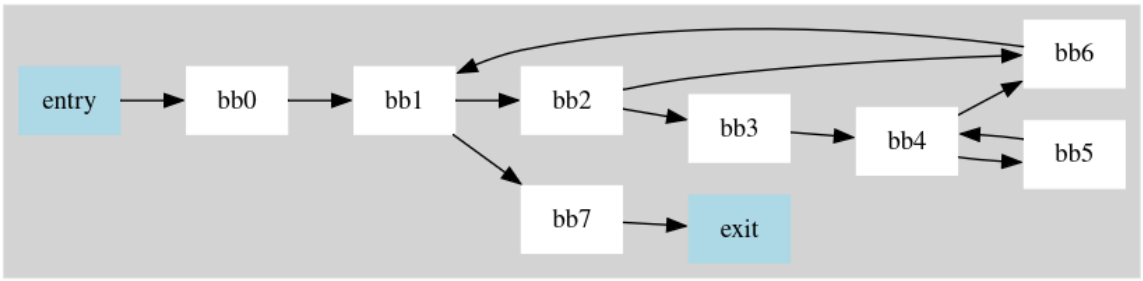

Рис. 1: CFG.

4 Дерево доминаторов.

Для данного графа построим дерево доминаторов.
Dom(bb0) = {bb1, bb2, bb3, bb4, bb5, bb6, bb7};
Dom(bb1) = {bb2, bb3, bb4, bb5, bb6, bb7};
Dom(bb2) = {bb3, bb4, bb5, bb6};
Dom(bb3) = {bb4, bb5};
Dom(bb4) = {bb5};
Dom(bb5) = ∅;
Dom(bb6) = ∅;
Dom(bb7) = ∅;
IDom(bb0) = ∅;
IDom(bb1) = bb0;
IDom(bb2) = bb1;

IDom(bb3) = bb2;
IDom(bb4) = bb3;
IDom(bb5) = bb4;
IDom(bb6) = bb2;
IDom(bb7) = bb1;

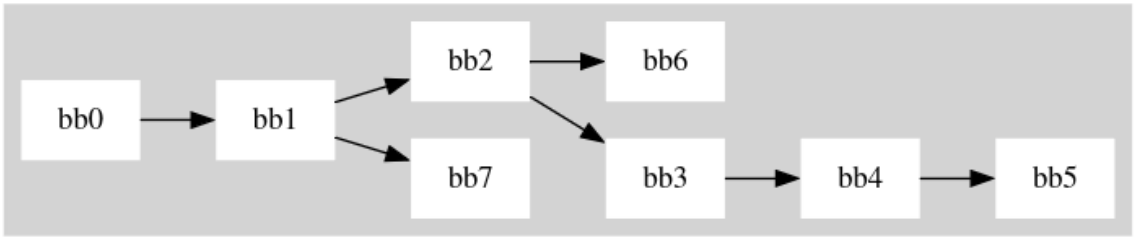

Рис. 2: DomG.

5 Граф фронта доминирования.

Для данного графа построим дерево доминаторов.
DF(bb0) = ∅;
DF(bb1) = ∅;
DF(bb2) = {bb1};
DF(bb3) = {bb6};
DF(bb4) = {bb4, bb6};
DF(bb5) = {bb4};
DF(bb6) = {bb1};
DF(bb7) = ∅;

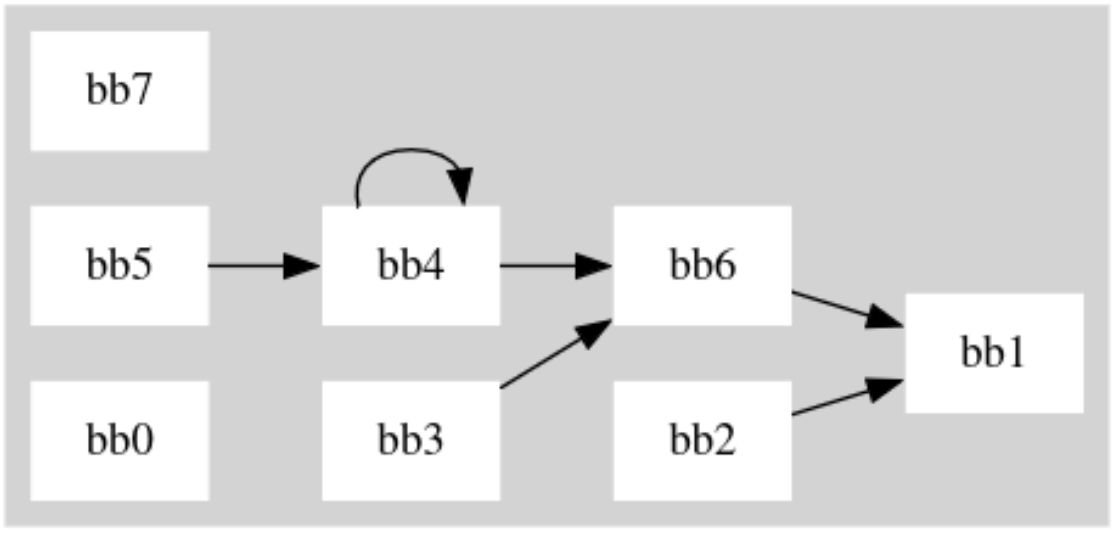

Рис. 3: граф DF.

6 Промежуточное представление Generic IR в SSA форме.

Переведем исходный Generic IR в SSA форму, добавив phi функции и переименовав переменные.

```
int function (int * a , int b , int c) {
    bb0:
        result0 = 0;
        i0 = b;
        goto bb1;
    bb1:
        result1 = phi(result0:bb0, result6:bb6);
        i1 = phi(i0:bb0, i2:bb6);
        if (i1 >= c)
            goto bb7;
        else
            goto bb2;
    bb2:
        x0 = 5;
        if (i1 % 4)
            goto bb6;
        else
            goto bb3;
    bb3:
        j0 = b;
        goto bb4;
    bb4:
        result2 = phi(result1:bb3, result5:bb5);
        j1 = phi(j0:bb3, j2:bb5);
        if (j1 > i1)
            goto bb6;
        else
            goto bb5;
    bb5:
        t0 = 2 * 3;
        addr0 = a + j1;
        val0 = *addr0;
        t1 = t0 * val0;
        t2 = t1 * x0;
        result3 = result2 + t2;
        result4 = result3 + 1;
        result5 = result4 + 2;
        j2 = j1 + 1;
        goto bb4;
    bb6:
        result6 = phi(result1:bb2, result2:bb4);
        i2 = i1 + 1;
        goto bb1;
    bb7:
        return result1;
}
```

7 Оптимизации.

7.1 Приведение цикла к каноническому виду.

Для применения оптимизаций приведем циклы к канонической форме. (В данной программе наблюдается три цикла: bb1->bb2->bb6->bb1/ bb1->bb2->bb3->bb4->bb6->bb1/ bb4->bb5->bb4;

для упрощения базовый блок 0 будет прехедером для первых двух циклов, а базовый блок 3 будет прехедером для третьего цикла).

```
int function (int * a , int b , int c) {
    bb0: //preheader1
        result0 = 0;
        i0 = b;
        if (i0 >= c)
            goto bb7;
        else
            goto bb1;
    bb1: //header1
        result1 = phi(result0:bb0, result6:bb6);
        i1 = phi(i0:bb0, i2:bb6);
        goto bb2;
    bb2: //latch1 (bb2 -> bb3 -> bb4 -> bb5 -> bb6)
        x0 = 5;
        if (i1 % 4)
            goto bb6;
        else
            goto bb3;
    bb3: //preheader2
        j0 = b;
        if (j0 > i1)
            goto bb6;
        else
            goto bb4;
    bb4: //header2
        result2 = phi(result1:bb3, result5:bb5);
        j1 = phi(j0:bb3, j2:bb5);
        goto bb5;
    bb5: //latch2
        t0 = 2 * 3;
        addr0 = a + j1;
        val0 = *addr0;
        t1 = t0 * val0;
        t2 = t1 * x0;
        result3 = result2 + t2;
        result4 = result3 + 1;
        result5 = result4 + 2;
        j2 = j1 + 1;
        if (j2 > i1)
            goto bb6;
        else
            goto bb4;
    bb6: //exiting2
        result6 = phi(result1:bb2, result2:bb4);
        i2 = i1 + 1;
        if (i2 >= c)
            goto bb7;
        else
            goto bb1;
    bb7: //exiting1
        return result1;
}
```

7.2 Цикловые оптимизации.

Для начала применим цикловые оптимизации. Граф сводимый и представлен в канонической LCSSA форме, поэтому мы можем применить оптимизации.

Рассмотрим цикл bb4->bb5->bb4. Предположим, что адреса вычисляются неэффективно и перейдем от базовой индуктивности j к новой базовой индуктивности addr.

```
bb3:
    addr0 = a + b;
    end0 = a + i1;
    if (addr0 > end0)
        goto bb6;
    else
        goto bb4;
bb4:
    result2 = phi(result1:bb3, result5:bb5);
    addr1 = phi(addr0:bb3, addr2:bb5);
    goto bb5;
bb5:
    t0 = 2 * 3;
    val0 = *addr1;
    t1 = t0 * val0;
    t2 = t1 * x0;
    result3 = result2 + t2;
    result4 = result3 + 1;
    result5 = result4 + 2;
    addr2 = addr1 + 1;
    if (addr2 > end0)
        goto bb6;
    else
        goto bb4;
```

Первый цикл является частью второго, а у второго цикла базовая индуктивность i; зависимых индуктивностей в данных циклах нет. Поэтому похожая оптимизация не прокатит.

7.3 Constant folding и алгебраическое переупорядочивание.

Вычислим выражения, аргументы которых являются константами, а также переставим некоторые выражения для упрощения следующих оптимизаций (Именения касаются только базового блока 5, остальные блоки остаются без изменения, поэтому покажем промежуточное представление только пятого блока).

```
bb5:
    val0 = *addr1;
    t0 = 6; // t0 = 2 * 3;
    t1 = t0 * x0;
    t2 = t1 * val0;
    result3 = result2 + t2;
    v0 = 3; // v0 = 1 + 2;
    result5 = result3 + v0;
    addr2 = addr1 + 1;
    if (addr2 > end0)
        goto bb6;
    else
        goto bb4;
```

7.4 Constant propagation.

Подставим в выражения значения аргументов, если те являются константами (Показаны только те блоки, в которых произошли изменения).

```
    bb2:
        // x0 = 5;
        if (i1 % 4)
            goto bb6;
        else
            goto bb3;
    bb5:
        val0 = *addr1;
        // to = 6;
        // t1 = t0 * x0 = 6 * 5 = 30;
        t2 = 30 * val0;
        result3 = result2 + t2;
        // v0 = 3;
        result5 = result3 + 3;
        addr2 = addr1 + 1;
        if (addr2 > end0)
            goto bb6;
        else
            goto bb4;
```

7.5 Результат.

```
int function (int * a , int b , int c) {
    bb0:
        result0 = 0;
        i0 = b;
        if (i0 >= c)
            goto bb7;
        else
            goto bb1;
    bb1:
        result1 = phi(result0:bb0, result6:bb6);
        i1 = phi(i0:bb0, i2:bb6);
        goto bb2;
    bb2:
        if (i1 % 4)
            goto bb6;
        else
            goto bb3;
    bb3:
        addr0 = a + b;
        end0 = a + i1;
        if (addr0 > end0)
            goto bb6;
        else
            goto bb4;
    bb4:
        result2 = phi(result1:bb3, result5:bb5);
        addr1 = phi(addr0:bb3, addr2:bb5);
        goto bb5;
    bb5:
        val0 = *addr1;
```

```
        t2 = 30 * val0;
        result3 = result2 + t2;
        result5 = result3 + 3;
        addr2 = addr1 + 1;
        if (addr2 > end0)
            goto bb6;
        else
            goto bb4;
    bb6:
        result6 = phi(result1:bb2, result2:bb4);
        i2 = i1 + 1;
        if (i2 >= c)
            goto bb7;
        else
            goto bb1;
    bb7:
        return result1;
}
```

7.6 Loop unroll и Loop Distribution.

Напоследок развернем цикл.

Программа примет следующий вид:

```
int function (int * a , int b , int c) {
    int result = 0;

    for ( int i = b / 4 ; i < c / 4 ; i++ ) {
        int k = 4 * i;
        for ( int j = b ; j <= k + 1 ; ++j )
            result += 3 ** (30 * a[j] + 3);
        result += 2 ** (30 * a[k + 2] + 3);
        result += 30 * a[k + 3] + 3;
    }
    if (b % 4 == 3) result -= 2 ** (30 * a[b - 1] + 3);

    int d = c / 4 * 4;
    if (c % 4 > 0) {
        for ( int j = b ; j <= d + 1 ; ++j )
            result += 30 * a[j] + 3;
    }
    if (c % 4 > 1) {
        for ( int j = b ; j <= d + 2 ; ++j )
            result += 30 * a[j] + 3;
    }
    if (c % 4 > 2) {
        for ( int j = b ; j <= d + 3 ; ++j )
            result += 30 * a[j] + 3;
    }
    return result;
}
```