Конструляторы

Основные определения

1. Промежуточные представления высокого, среднего, низкого уровня.
2. Базовый блок.
3. Граф потока управления.
4. Локальная оптимизация.
5. Глобальная оптимизация.
6. Ориентированный ациклический граф.
7. Остовное дерево.
8. Поток данных.
9. Точка программы.
10. Состояние программы.
11. Передаточная функция инструкции.
12. Передаточная функция базового блока.
13. Определение переменной.
14. Использование переменной.
15. Достигающее определение.
16. Живая переменная.
17. Доступное выражение.
18. Избыточные вычисления.
19. Полурешетка с операцией «сбор».
20. Верхний и нижний элементы полурешетки.
21. Наибольшая нижняя граница двух элементов полурешетки.
22. Диаграмма полурешетки.
23. Структура потока данных.
24. Монотонная структура анализа потока данных.
25. Дистрибутивная структура анализа потока данных.
26. Фиксированная точка системы уравнений.
27. Максимальная фиксированная точка системы уравнений.
28. Решение сбором по всем выполнимым путям.
29. Решение сбором по всем путям.
30. Консервативность анализа.
31. Доминатор.
32. Дерево доминаторов.
33. Постдоминатор.
34. Дерево постдоминаторов.
35. Граница доминирования.
36. Обратная граница доминирования.
37. Зависимость по данным.
38. Зависимость по управлению.
39. Эквивалентность по управлению.
40. Сворачивание констант.
41. Распространение копий.
42. Остовное ребро.
43. Прямое ребро.
44. Обратно направленное ребро.
45. Обратное ребро.
46. Естественный цикл.
47. Инвариант цикла.
48. 48.Инструкция, инвариантная относительно цикла.
49. Индуктивная переменная.
50. Семейство индуктивных переменных.
51. Частичная раскрутка циклов.
52. Гнездо циклов.
53. Выравнивание циклов.
54. Слияние циклов.
55. Бесполезный код.
56. Недостижимый код.
57. Форма статического единственного присваивания (*SSA*-форма).
58. Фи-функция.
59. Максимальная *SSA*-форма.
60. Частично усеченная *SSA*-форма.
61. Критическое ребро.

62. Суперблок.
63. Область графа потока управления.
64. Дерево управления.
65. Сводимость ГПУ
66. Профилирование.
67. 67.Инструментирование и семплирование.
68. Межпроцедурный анализ.
69. Граф вызовов.
70. Контекст.
71. Чувствительность к контексту.
72. Функции скачка.
73. Машинно-ориентированная оптимизация.
74. Режимы адресации.
75. Выбор команд.
76. Переписывание дерева.
77. Что такое распределение регистров.
78. Интервалы жизни переменных.
79. Расщепление интервалов жизни.
80. Слияние интервалов жизни.
81. Граф конфликтов.
82. Слив интервалов жизни.
83. Планирование кода.
84. Граф зависимостей по данным.
85. Граф зависимостей программы.
86. Топологический порядок.
87. Топологическая сортировка.
88. Программная конвейеризация.
89. Ресурсы. Виды ресурсов.
90. Таблица резервирования ресурсов.
91. Пространственная локальность данных.
92. Временная локальность данных.
93. Для чего необходимо обеспечивать локальность данных?
94. Распространение копий.
95. Распространение констант.

Теормин

1. Промежуточное представление высокого, среднего и низкого уровня (HIR, MIR, LIR).

(HIR) – атрибутированное абстрактное синтаксическое дерево

MIR - Middle-level Intermediate Representation. Включает в себя:

- **трёхадресный код** (трёхадресные инструкции, “четвёрки”)
  - инструкции присваивания (`x <- op y z` и вариации)
  - инструкции перехода: `goto, ifTrue, ifFalse` и пр.
  - процедуры: `call, return, param` (передача параметра вызываемой функции)

5 таблицу символов

- переменные, их имена в программе и атрибуты, такие как область видимости, тип, для имен функций — число параметров, и т. п.

LIR - биткод LLVM

LIR используется для машинно-зависимых задач вроде распределения регистров и выбора подходящих команд.

2. Базовый блок.

Базовым блоком (**ББ** или линейным участком)
называется последовательность следующих одна за другой инструкций MIR, обладающая следующими свойствами:

1. поток управления может входить в базовый блок только через его первую инструкцию (т.е. в программе нет переходов в средину базового блока);
2. поток управления покидает базовый блок без останова или ветвления, кроме, возможно, последней инструкции базового блока.

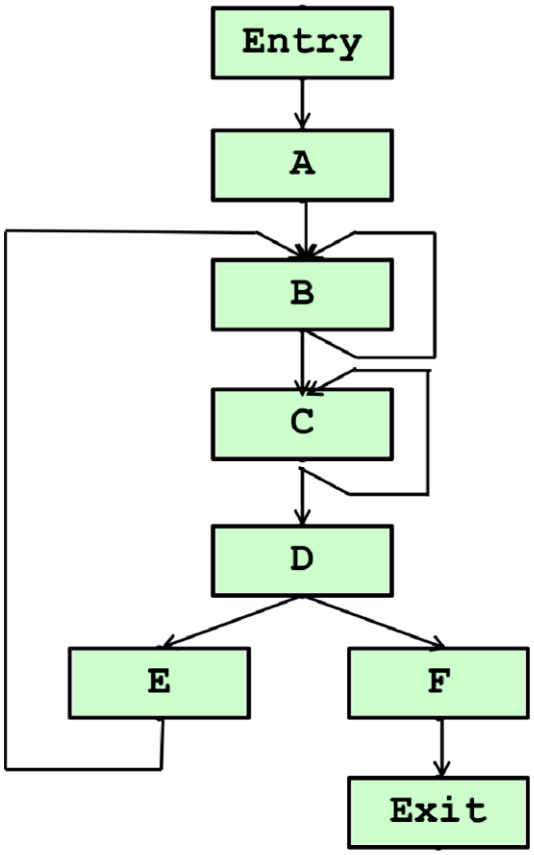
Начало базового блока (НББ), таким образом, одно из:

- первая инструкция программы;
- помеченная инструкция программы;
- инструкция, следующая за инструкцией перехода

3. Граф потока управления.

Граф потока управления (ГПУ):

- вершины графа — базовые блоки
- дуга соединяет выход одного блока со входом другого, если второй блок может выполняться сразу следом за первым.
    - Например, если последней инструкцией базового блока является инструкция условного перехода, то из этого блока будут выходить две дуги.
    - А если первая инструкция базового блока имеет метку, то в этот блок будут входить дуги из всех базовых блоков, последняя инструкция которых будет инструкцией условного или безусловного перехода на эту метку.



4. Локальная оптимизация.

Оптимизация - выполнение следующих преобразований:

- Удаление общих выражений (инструкций, повторно вычисляющих уже вычисленные значения)
- Удаление мертвого кода (инструкций, вычисляющих значения, которые впоследствии не используются)
    - Сворачивание констант (выражение 2 * 3.14 можно заменить значением 6.28)
- Изменение порядка инструкций, там, где это возможно (чтобы сократить время хранения временного значения на регистре)
- Локальное снижение стоимости вычислений, т.е. замена более дорогих операций более дешевыми (например, 2x -> x+x, x/2 -> x*0.5)

Все вышеперечисленные преобразования можно выполнить за один просмотр базового блока, представив его в виде ориентированного ациклического графа

5. Глобальная оптимизация.

Оптимизация в пределах процедуры, которая учитывает потоки данных между базовыми блоками

6. **Ориентированный ациклический граф** – ориентированный граф, в котором отсутствуют направленные циклы, но могут быть параллельные пути.
7. **Остовное дерево** – остовное дерево графа состоит из минимального подмножества ребер графа, таких, что из одной вершины графа можно попасть в любую другую вершину двигаясь по этим ребрам

8. **Поток данных** – множество состояний на входе и выходе в базовый блок
9. **Точка программы** – точки программы расположены между инструкциями программы.
10. **Состояние программы** – множество значений всех пременных включая значения переменных в стеке.
11. **Передаточная функция инструкции.** Состояния во входной и выходной точках фрагмента определяет передаточные функции этого фрагмента: передаточная функция прямого обхода (f) переводит состояние во входной точке в состояние в выходной точке, а передаточная функция обратного обхода (f b) переводит состояние в выходной точке в состояние во входной точке
12. **Передаточная функция базового блока.** Передаточная функция fB блока B равна композиции передаточных функций его инструкций

13. Определение переменной.

Определением переменной x называется инструкция, которая присваивает значение переменной x

Каждое определение переменной x убивает все другие её определения

Каждая переменная может иметь несколько определений

14. Использование переменной.

Использованием переменной x является инструкция, одним из операндов которой является переменная x.

15. Достигающее определение.

Определение d достигает точки p, если существует путь от точки, непосредственно следующей за d, к точке p, такой, что вдоль этого пути d не убивается

16. Живые переменные.

Живыми называются переменные, значения которых, вычисленные в рассматриваемом базовом блоке, используются в других базовых блоках

17. Доступное выражение.

Выражение *e* доступно в точке *p*, если *e* вычисляется на любом пути от Entry до *p*, причём переменные, входящие в состав *e*, не переопределяются между последним таким вычислением и точкой *p*.

18. **Избыточные вычисления –** повторный или неоднократный пересчет одного и того же.

19. Полурешётка.

Полурешётка – это абстрактная алгебраическая структура, над элементами которой определена абстрактная операция «сбор», обладающая свойствами операций пересечения и объединения. (Пояснение: полурешетки используются для обобщения алгоритмов анализа потока данных.)

Операция «сбор».

Операция $\wedge$, определенная на множестве L(полурешётке), такая что для всех $x, y, z \in L$:

- $x \wedge x = x$ - идемпотентность
- $x \wedge y = y \wedge x$ - коммутативность
- $x \wedge (y \wedge z) = (x \wedge y) \wedge z$ - ассоциативность

20. Верхний и нижний элемент полурешётки.

Верхний элемент (верх) $T \in L$ такой, что для всех $x \in L$ выполняется $T \wedge x = x$

Нижний элемент $T \in L$ такой, что для всех $x \in L$ выполняется $T \wedge x = T$

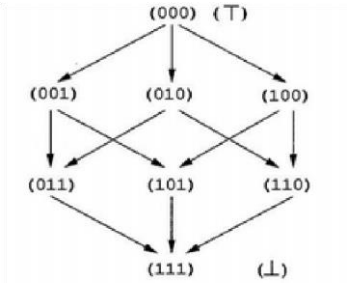
21. Наибольшая нижняя граница двух элементов полурешётки.

Наибольшей нижней границей $inf(x,y)$ элементов $x,y \in L$ называется элемент $g \in L$, такой, что $g \leq x$, $g \leq y$ и если $z \in L$, такой, что $z \leq x$ и $z \leq y$, то $z \leq g$

Утверждение: Если x и $y \in L$, где $< L, \wedge >$ – полурешётка, то $inf(x,y) = x \wedge y$

22. Диаграмма полурешётки.

Представляет собой граф, узлами которого являются элементы L, а рёбра направлены от x к y, если y <= x.



Пример для $< U, \cup >$, $|U| = 8$, элемент множества U представляется битовым 3-вектором.

23. Структура потока данных. Называется четвёрка $< D, F, L, \wedge >$, где D — направление анализа (Forward/Backward), F — семейство передаточных функций, L — поток данных, $\wedge$— операция сбора.

Семейство передаточных функций называется замкнутым, если:

1) F — содержит тождественную функцию $I: \forall x \in L: I(x) = x$
2) F — замкнуто относительно композиции $\forall f, g \in F \Rightarrow h(x) = g(f(x)) \in F$

24. Монотонная структура потока данных. Структура потока данных $< D, F, L, \wedge >$ называется монотонной, если: $\forall x, y \in L, \forall f \in F \quad f(x \wedge y) \leq f(x) \wedge f(y)$

Или:

$$\forall x, y \in L, \forall f \in F \quad x \leq y \Rightarrow f(x) \leq f(y)$$

25. Дистрибутивная структура потока данных.

Структура потока данных $< D, F, L, \wedge >$ называется дистрибутивной, если: $\forall x, y \in L, \forall f \in F f(x \wedge y) = f(x) \wedge f(y)$

Если дистрибутивна => монотонна

26. Фиксированная точка системы уравнений

фиксированная точка (FP) системы уравнений
$In[B] = \wedge f_p(In[P]), p \in Pred(B)$ представляет собой решение этой системы.

27. Максимальная фиксированная точка.

Максимальная фиксированная точка (MFP) системы уравнений

$In[B] = \wedge f_p(In[P]), p \in Pred(B)$ представляет собой решение $In[Bi]^{max}$этой системы, обладающее тем свойством, что для любого другого решения $In[Bi]$ выполняется $In[Bi] \leq In[Bi]^{max}$

28. Решение сбором по всем выполнимым путям

Идеальным решением системы уравнений потока данных для In[Bk] будет: Ideal[Bk]=∧p∈P fp(In[lentry]), P={P1,P2,...} —множество всех выполнимых путей от Entry до Bk. Путь P является выполнимым тогда и только тогда, когда известно выполнение программы (начальные данные), которое следует в точности по этому пути.
Решение названо идеальным, так как:
1) оно наиболее точное

2) вычислить его почти никогда не удаётся, так как поиск всех возможных путей выполнения — задача неразрешимая

29. Решение сбором по всем путям.

Решение сбором по всем путям (MOP-решение) от Entry до входа в Bk определяется соотношением $MOP[Bk] = \wedge_{p \in Q} f_p(I_{entry})$, где Q = {P1, P2, ...} — множество всех путей от Entry до входа в Bk. Пути, рассматриваемые в MOP-решении — это надмножество всех выполнимых путей: MOP-решение собирает значения потоков данных как для всех выполнимых путей, так и для путей, которые не могут быть выполнены. Следовательно, для всех Bk выполняется соотношение $MOP[Bk] \leq Ideal[Bk]$

30. Консервативность.

Решение FP системы уравнений потока данных называется консервативным, если выполняется условие: FP ≤ MFP.

(Другой вариант, но, кажется, не по этой теме: консервативность — это когда компилятор обязан считать, что две команды могут обращаться к одним и тем же ячейкам, если он не может доказать обратное).

31. Доминатор.

В ГПУ вершина d является доминатором вершины n (d dom n или d = Dom(n)), если любой путь от вершины Entry до вершины n проходит через вершину d. Каждая вершина n является доминатором самой себя.

32. Постдоминатор.

В ГПУ вершина p постдоминирует над вершиной n (p = Postdom(n)), если каждый путь из вершины n в вершину Exit проходит через вершину p. Постдоминаторы ГПУ — это доминаторы его обратного графа.

33. Дерево доминаторов.

Если соединить дугами каждый блок с его непосредственным доминатором, получим дерево доминаторов.

34. Дерево постдоминаторов (аналогично с 33)

35. Граница доминирования.

Граница доминирования n (DF(n)) — множество узлов m, удовлетворяющих условиям:

1) n является доминатором предшественника m, то есть
   a) $p \in Pred(m)$

b) $n \in Dom(p)$

2) n не является строгим доминатором m, то есть

a) $n \notin (Dom(m) - \{m\})$

Короче говоря, граница доминирования — это множество первых узлов, которые достижимы из n, на любом пути графа потока, проходящем через n, но над которыми n не доминирует

36. Обратная граница доминирования.

RDF(n) — граница доминирования вершины n в обратном графе.

Обратный граф — граф, у которого направления всех рёбер противоположны.

37. Зависимость по данным

Две команды называются зависимыми по данным, если изменение порядка их выполнения может привести к изменению результата вычислений, выполняемых программой.
Виды зависимостей:
Истинная зависимость: чтение после записи. Если команда C1 записывает значение в некоторую ячейку памяти, а команда C2 считывает это значение, то команды C1 и C2 зависимы.

Антизависимость: запись после чтения. Если команда C1 считывает значение из некоторой ячейки памяти, а команда C2 записывает в эту ячейку новое значение , то команды C1 и C2 зависимы.

Зависимость по выходу: запись после записи. Если команды C1 и C2 записывают значения в одну и ту же ячейку памяти, то команды C1 и C2 зависимы

38. Зависимость по управлению.

Команда s1 зависит по управлению от команды s2, если *результат* выполнения команды s2 определяет, будет ли выполняться команда s1. из темы 4(36 слайд):

По определению, вершина m ГПУ зависит по управлению от вершины n тогда, и только тогда, когда: существует непустой путь T от n до m, такой что для любых k из (T –{n}): m = Postdom(k), т.е. если выполнение программы пошло по пути T, то, чтобы достичь exit, оно обязательно пройдет через m. m не обязательно является строгим постдоминатором n: у n может быть несколько выходов, так что помимо T возможны и другие пути, проходящие через n, но потом ведущие не в m, а в другие вершины.

39. Эквивалентность по управлению.

Два базовых блока Bi и Bj эквивалентны по управлению, если Bi выполняется тогда и только тогда, когда выполняется Bj.

40. Сворачивание констант.

Заключается в вычислении констант в процессе компиляции и замене константных выражений их значениями. Например 2 * 3.14 заменить на 6.28.

41. Распространение копий.

Пусть есть инструкция копирования $x \leftarrow y$. Распространение копий означает замену всех последующих вхождений переменной x на переменную y.

42. Остовная дуга ГПУ.

Дуги ГПУ, являющиеся дугами его глубинного остовного дерева (оно строится однозначно, согласно алгоритму).

43. Прямая дуга ГПУ.

Дуги ГПУ, не являющиеся дугами его остовного дерева, но имеющие такое же направление, что и остовные дуги

44. Обратно направленная дуга ГПУ.

Дуги ГПУ, направленные противоположно остовным

45. Обратная дуга ГПУ.

Обратно направленная дуга ГПУ $< B_i, B_k >$ называется обратной, если $B_k = Dom(B_i)$

46. Естественный цикл (натуральный).

Цикл со следующими свойствами:

1) имеет единственный входной узел, называемый заголовком
2) существует обратное ребро, ведущее в заголовок цикла

Естественный цикл обратного ребра (Bi,Bk) составляют узел Bk(заголовок цикла) и все узлы ГПУ, из которых можно достичь узла Bi, не проходя через узел Bk.(эти узлы составляют тело цикла).

47. Инвариант цикла

- CONST C
- Переменная u, все определения которой находятся вне цикла
- Переменная v, определенная внутри цикла, если:
    - У v есть всего одно определение
    - Это определение находится в ББ, являющимся доминатором всех выходов из цикла

48. Инструкция, инвариантная относительно цикла.

Если она удовлетворяет одному из условий:-

1) её операнды — константы

2) все определения операндов, достигающие инструкции, находятся вне цикла
3) внутри цикла имеется в точности одно определение операнда, но оно само инвариантно относительно цикла

49. Индуктивная переменная.

Определение. Переменная v называется индуктивной переменной цикла L, если на каждой итерации цикла значение v увеличивается на значение переменной (или константы) c, являющейся инвариантом цикла, то есть на каждом витке цикла выполняется инструкция: v <- +, v, c

50. Семейство индуктивных переменных.

Основной индуктивной переменной по определению является индуктивная переменная i, все определения которой эквивалентны определению вида i <- +, i, c, где c – инвариант цикла (как правило, константа). Нет необходимости, чтобы значение c было одинаковым в каждом таком определении. Основная индуктивная переменная является линейной, если в цикле имеется всего одно ее определение, причем это определение является доминатором (или постдоминатором) всех остальных вершин цикла. - Производная индуктивная переменная j выражается через одну из основных индуктивных переменных i как c * i + d, где c и d – инварианты цикла. - Основная индуктивная переменная i и все ее производные индуктивные переменные составляют семейство индуктивных переменных.

51. Частичная раскрутка циклов.

Раскрутка цикла копирует тело цикла для нескольких итераций и корректирует вычисление индексов соответствующим образом.

52. Гнездо циклов.

Гнездо циклов - набор циклов, расположенных таким образом, что один цикл является телом другого цикла, гнездо циклов может состоять из одного цикла.

53. Выравнивание циклов.

Если ni не делится на m, то остается кусок цикла, который выполняется в коротком цикле-прологе. Назначение цикла-пролога – гарантировать, что количество итераций цикла раскручиваемого на m, кратно m. Такое преобразование цикла (расщепление цикла для обеспечения удобных границ) называется выравниванием цикла.

54. Слияние циклов.

Слияние циклов – объединение двух циклов с одинаковыми границами изменения индекса и шагом в один.

55. Бесполезный код.

Инструкции, которые не влияют на результат вычислений

56. Недостижимый код.

Инструкции, которые не содержатся ни в одном реальном пути выполнения

57. SSA-форма.

Форма статического единственного присваивания, которая позволяет в каждой точке программы объединить информацию об имени переменной с информацией о текущем значении этой переменной (или, что то же самое, с информацией о том, какое из определений данной переменной определяет её текущее значение в данной точке).
Хотелось бы, чтобы программа в SSA-форме удовлетворяла условиям:

1) каждое определение переменной имеет индивидуальное имя
2) каждое использование переменной достигало только одно определение

58. Что такое фи-функция.

Пояснение: если в рассматриваемый блок можно прийти несколькими путями, то на этих путях могут быть разные определения одной и той же переменной. Каждому входному ребру рассматриваемого блока соответствует своя фи-функция, которая нужна для того, чтобы для рассматриваемого блока сопоставить значение входных параметров с их текущими значениями.

Формально: фи-функция — это функция объединения значений. Фи-функция определяет SSA имя для значения своего аргумента, соответствующего ребру, по которому управление входит в блок. При входе в базовый блок все его фи-функции выполняются одновременно и до любого другого оператора, определяя целевые SSA-имена.

59. Максимальная SSA-форма

вставляем ϕ-функцию после каждой точки сбора. содержит больше ϕ функций, чем надо

60. Частично-усечённая SSA-форма.

SSA-форма, построенная с помощью алгоритма, использующего DF(границу доминирования), то есть размещение фифункций только в границу доминирования. Содержит меньше фи-функций, чем обычная SSA форма (не минимальное количество).

61. Критическая дуга.

По определению ребро, выходящее из вершины с несколькими потомками, и входящее в вершину с несколькими предками, называется критическим. Критические ребра обычно исключают, разрывая их и вставляя в разрыв дополнительные вершины.

62. Суперблок.

Расширенный базовый блок E определяется как множество базовых блоков B1, B2, …, Bn, где у блока B1 может быть несколько предшественников, а каждый из остальных блоков имеет в суперблоке единственного предшественника. Блоки Bi формируют дерево с корнем в B1. У супер блока E может быть несколько выходов на блоки (или суперблоки) не входящие в состав E.

63. Область.

Областью графа потока называется его подграф $R = \langle N_R, E_R \rangle$ такой, что:

1) существует узел $h \in N_R$, доминирующий над всеми узлами в R; этот узел называется заголовком области R;
2) если из некоторого узла r_2 ГПУ можно достичь узла $r_1 \in R$, минуя заголовок h, то и $r_2 \in R$;
3) множество E_R включает все ребра ГПУ между любыми узлами $r_1, r_2 \in R$, за исключением некоторых рёбер, входящих в заголовок h;
4) единственным входом в область является её заголовок

Область-лист.

Каждый базовый блок B может рассматриваться как область $R = \langle \{B\}, \varnothing \rangle$. Такая область называется область-лист.

Область-тело.

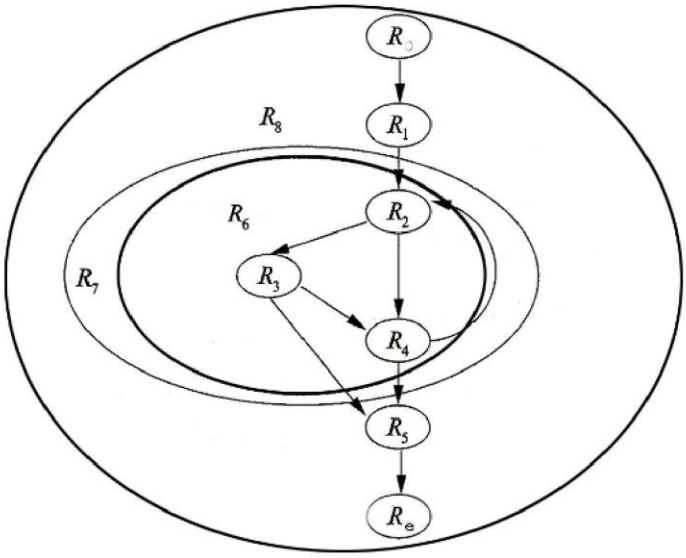
Пусть L – самый внутренний цикл гнезда циклов. Тело цикла L (все узлы и ребра, за исключением обратных ребер к заголовку цикла) можно заменить узлом, представляющим область R. Такая область называется область-тело.

Область-цикл.

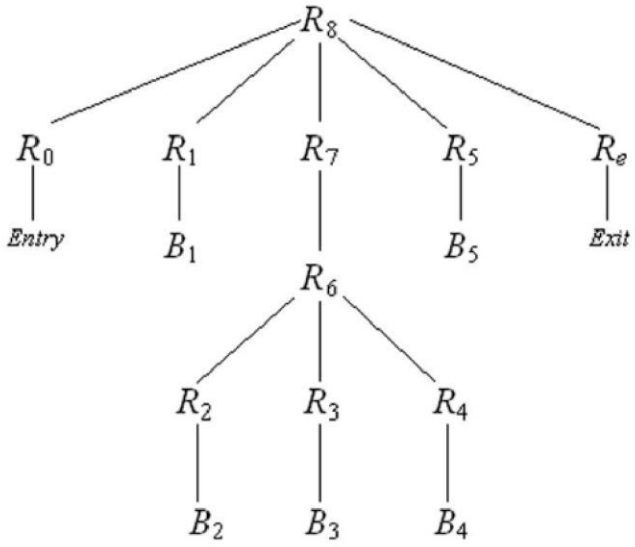
Если к области-телу R, соответствующей телу цикла L присоединить обратное ребро к заголовку цикла L, получится новая область Q. Такая область называется область-цикл.

64. Дерево управления.

Иерархия всех областей в виде дерева.



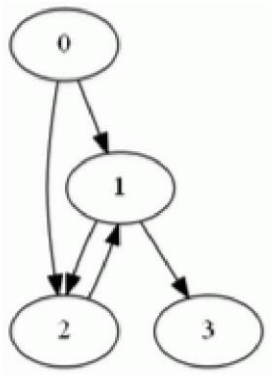
Вся иерархия



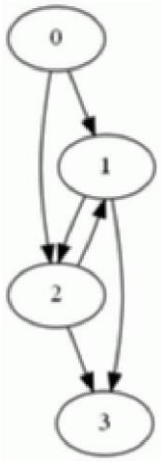
Дерево управления

65. Сводимость ГПУ.

ГПУ называется сводимым, если для него можно построить дерево управления. Примеры несводимых графов:



Пример с GOTO



Классический пример

9.1.4. Алгоритм построения иерархии областей

♢ **Алгоритм**.

9. Найти все естественные циклы.
2. Найденные естественные циклы упорядочить изнутри гнезд наружу, т.е. начиная с наиболее внутренних циклов.
3. Выбрать очередной естественный цикл (сначала – самый первый, потом – следующий по порядку).
   Если циклов больше нет, **алгоритм заканчивается**.
4. Тело выбранного цикла L (все узлы и ребра, за исключением обратных ребер к заголовку) заместить узлом R, представляющим область-тело.
   После замещения обратное ребро в заголовок L становится петлей.
5. Построить область-цикл Q, представляющую цикл L (в отличие от области R область Q не содержит петли).
   Перейти к шагу 3.

20

Вход: приводимый ГПУ G.

Выход: список областей графа G, который может использоваться в задачах анализа потоков данных на основе областей.

Метод: вышеописанный цикл

Построение областей и дерева управления позволяет ускорить обход графа потока управления во время выполнения различных алгоритмов анализа потока данных. В алгоритмах анализа потока данных, в которых в качестве сбора используется объединение, удается, заменив каждый цикл узлом типа область-цикл, оставить только ациклические пути для распространения атрибутов. Это позволяет сократить время выполнения соответствующего анализа потока данных.

66. Профилирование.

Профилирование – это численная оценка времени выполнения каждого фрагмента (базового блока, области) процедуры. В результате профилирования получается временной, или частотный, или ещё какой-нибудь профиль процедуры.

67. Семплирование

Прерываем (с помощью прерываний) с некоторой периодичностью программу, анализируем состояние программы, и строим на основе этого статистическую модель.

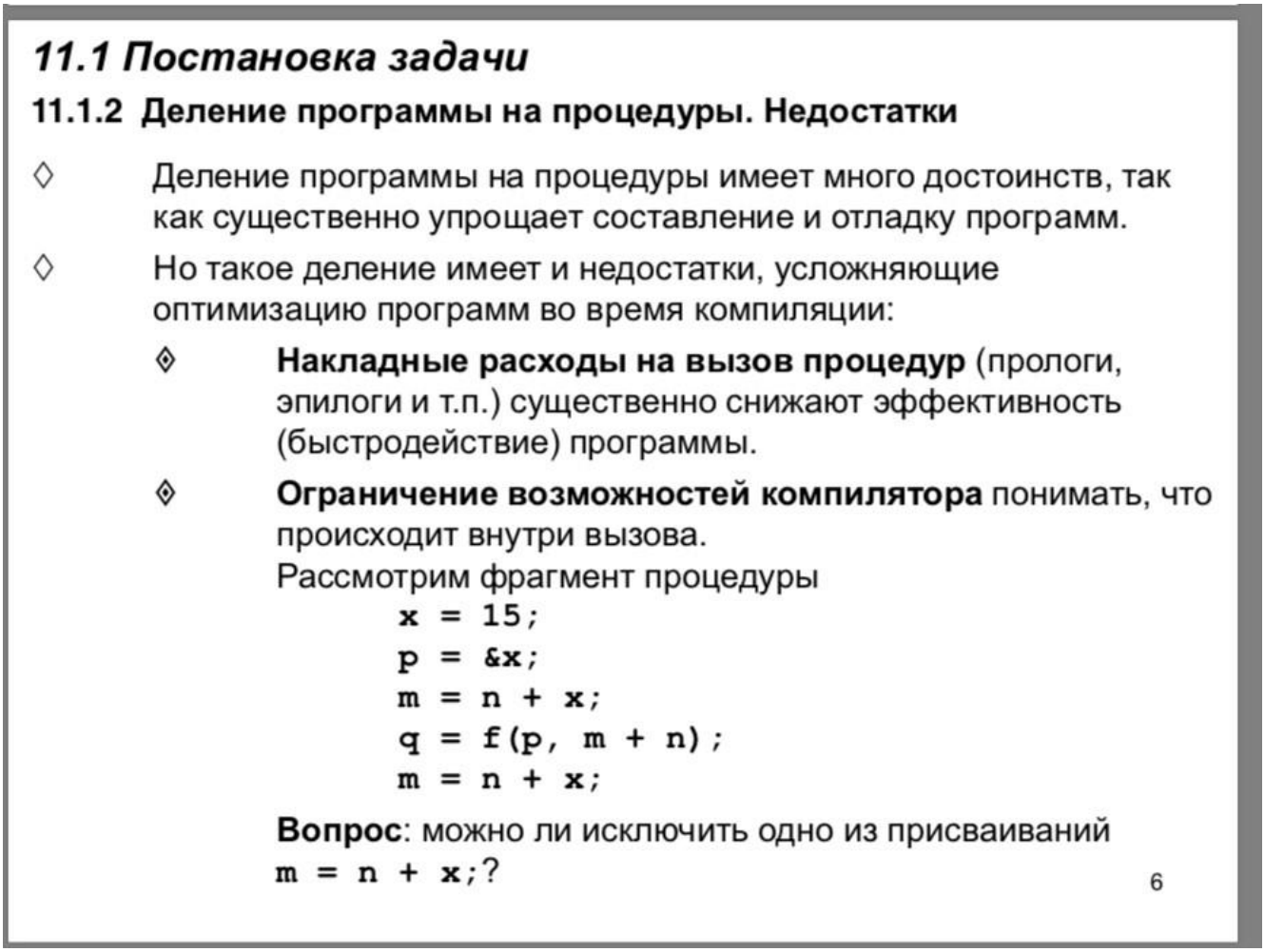
Бывает программное и аппаратное.

Плюсы: код не изменяется, меньше накладных расходов

Инструментирование.

Инструментирование – в различные точки процедуры вставляются вызовы соответствующего инструмента (счетчика количества обращений к фрагменту, секундомера и т.п.), выбираются наборы исходных данных, обеспечивающие достаточно полное покрытие процедуры и исследуемая процедура запускается на этих наборах, в результате чего получается требуемый профиль.

68. Межпроцессуальный анализ



11.1 Постановка задачи

11.1.2 Деление программы на процедуры. Недостатки

- ◊ Деление программы на процедуры имеет много достоинств, так как существенно упрощает составление и отладку программ.
- ◊ Но такое деление имеет и недостатки, усложняющие оптимизацию программ во время компиляции:
  - ◈ **Накладные расходы на вызов процедур** (прологи, эпилоги и т.п.) существенно снижают эффективность (быстродействие) программы.
  - ◈ **Ограничение возможностей компилятора** понимать, что происходит внутри вызова.
    Рассмотрим фрагмент процедуры

```
x = 15;
p = &x;
m = n + x;
q = f(p, m + n);
m = n + x;
```

Вопрос: можно ли исключить одно из присваиваний `m = n + x;`?

6

69. Граф вызовов.

Взвешенный граф вызовов (ВГВ) задается пятеркой

$G = <N, p_N, E, p_E, main>$

N – множество вершин (представляющих процедуры программы), с каждой вершиной связан ее «вес» p_N – целое число, равное количеству инструкций соответствующей

процедуры

E – множество ребер (каждое ребро соединяет вызываемую и вызывающую процедуры), с

каждым ребром связан его «вес» p_E – количество выполнений соответствующего вызова

main – имя процедуры, выполняемой первой.

70. Контекст.

Эт тип когда переменные в функции (глобальные, параметры), которые в ней используются, во время разных вызовов могут принимать разные значения. За счет этого функция будет вести себя по-другому. (см. Тема 11, слайды 7+)

71. Контекстная чувствительность.

Вызывая одну и ту же функцию в разных местах она будет вести себя по-другому. (см. Тема 11, слайды 14+)

72. Функция скачка

Во время обработки каждой вершины графа вызовов анализатор должен определять как константные значения, известные на входе в процедуру, влияют на множество константных значений в каждой точке вызова. Для этого он строит небольшую модель, называемую *функцией скачка*, для каждого фактического параметра.

Точке вызова s с n параметрами соответствует вектор функций скачка

$$\mathcal{J}_s = \left(\mathcal{J}_s^{p_1}, \mathcal{J}_s^{p_2}, \ldots, \mathcal{J}_s^{p_n}\right)$$

где p_i – фактические параметры вызова (i = 1,2,…,n).

Каждая функция скачка $\mathcal{J}_s^{p_i}$ зависит от некоторого подмножества формальных параметров $Support\left(\mathcal{J}_s^{p_i}\right)$

Требуется, чтобы $\mathcal{J}_s^{p_i}$ возвращала значение $Undef$, если хотя бы один параметр из $Support\left(\mathcal{J}_s^{p_i}\right)$ имеет значение $Undef$.

39

73. **Машинно-ориентированная оптимизация.**

- учет регистровой структуры вычислительной аппаратуры (оптимальное
- распределение регистров, передача параметров в процедуры и функции через регистры)
- удаление лишних команд
- оптимизация потока управления и удаление недостижимых участков программ
- снижение "стоимости" программы (есть «дорогие» и «дешевые» команды с точки зрения времени их выполнения процессором)
- использование «машинных идиом» (например: xor ax,ax для обнуления регистра)
- слияние, дробление и развертывание циклов, иногда требующееся из-за технических особенностей аппаратуры

- учет векторных и конвейерных свойств архитектуры

74. **Режимы адресации.**

-Режим прямой адресации – адрес задается непосредственно значением x

-Режим индексированной адресации a(r), a – переменная, r – регистр. Адрес a(r) вычисляется путем прибавления к l -значению a значения из регистра r. Этот режим адресации полезен для обращения к массивам: a – базовый адрес массива, r – смещение.

-Режим косвенной адресации: r ссылка на ячейку памяти, находящуюся по адресу contents(r)*c(r) ссылка на ячейку памяти, находящуюся по адресу c + contents(r) c – константа, contents – операция разыменования)

-Режим адресации с использованием констант, для указания которых используется префикс #. Например, команда LD r, #n загружает в регистр r целое число n

75. Выбор команд.

Отображение инструкций промежуточного представления в команды целевого процессора называется выбором команд.

76. Переписывание дерева

13.5.2 Описание метода

- ◊ Целевой код генерируется в процессе *свертки входного дерева в единый узел* путем последовательного применения *правил преобразования дерева*.
- ◊ Каждое *правило преобразования дерева* представляет собой инструкцию вида

$$replacement \leftarrow template\ \{action\}$$

где *replacement* – отдельный узел, *template* – шаблон (поддерево), *action* – действие (фрагмент генерируемого кода, соответствующий шаблону).

- ◊ Множество правил преобразования дерева именуется *схемой трансляции дерева*.
- ◊ Трансляция состоит из (возможно, пустой) последовательности машинных команд, которые генерируются действием (*action*), связанным с шаблоном (*template*).

22

77. **Что такое распределение регистров.**

Распределение регистров отображает неограниченное множество имен (псевдорегистров) на конечное множество физических регистров целевой машины (NP-полная задача).

78. Интервал жизни.

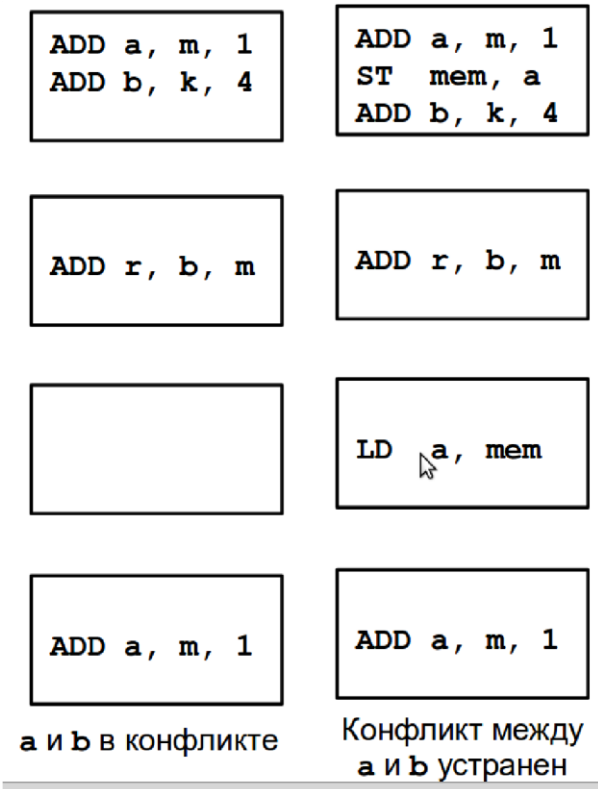
Интервалом жизни (ИЖ) значения w переменной v называется множество команд программы, начиная с команды, в которой переменная v определяется со значением w, и кончая последней командой, в которой переменная v используется с этим значением.

79. Слияние интервалов жизни.

Рассмотрим команду копирования LRi = LRj. Если ИЖ LRi и LRj не находятся в конфликте (не являются смежными узлами ГК), то команду можно исключить и все ссылки на LRj заменить ссылками на LRi. В результате ИЖ LRi и LRj как бы сольются.

a = b и они не находятся в конфликте => заменяем все b на a

80. Расщепление интервалов жизни.



- На левом рисунке LR_a и LR_b полностью пересекаются. Расщепив LR_a на два LR, удается устранить конфликт.
- При этом команды **ST** и **LD** применяются только здесь, а не перед всеми **a**.

81. Граф конфликов

граф у которого вершины - интервалы жизни, а рёбра соединяют те интервалы жизни, которые находятся в конфликте

Два интервала жизни находятся в конфликте, если один из них определяется в месте, где актуален второй

82. Слив интервалов жизни

14.3 Глобальное распределение и назначение регистров

14.3.13 Слив переменных

- *Слив* временной переменной **t** состоит в следующем:
  - в «автоматической» памяти процедуры выделить ячейку **ta** для хранения значений **t** (эту ячейку называют «дом» **t**);
  - перед каждой командой, использующей **t**, вставить команду **LD t, ta**;
  - после каждой команды, определяющей **t**, вставить команду **ST ta, t**;
  - просматривают текст полученной процедуры с целью выявить и удалить лишние команды **LD** и **ST** (иногда это удается).

83. Планирование кода.

Цель планирования кода – выбрать такую последовательность команд, которая, не меняя семантики программы, обеспечит близкое к оптимальному использование особенностей архитектуры целевого процессора

Прежде всего необходимо обеспечить использование возможностей параллельного выполнения команд, реализованных в аппаратуре

84. **Граф зависимостей по данным.**

Строится граф зависимостей по данным G = (N, E)

- N — вершины графа, соответствующие каждой операции внутри ББ.

Для каждой n ∈ N определяется таблица

резервирования ресурсов RTn

- E — рёбра графа, соответствующие зависимостям по данным между операциями.

Для каждого ребра e=(u,v) ∈ E определяется

задержка de, такая, что операция v должна

выполниться не ранее, чем через de после

выполнения операции u

(Если за операцией n1 идет операция n2 и они обращаются к одной и той же ячейке памяти с запаздыванием l1 и l2 соответственно, то нужно добавить ребро (n1, n2) с de=l1- l2)

85. Граф зависимостей программы.

Граф, узлами которого являются инструкции присваивания программы, а ребра указывают зависимости данных между инструкциями и их направлениями. Ребро от инструкции s1 к инструкции s2 имеется в том и только том случае, когда имеется зависимость данных между некоторым динамическим экземпляром s1 и более поздним динамическим экземпляром s2.

86. Топологический порядок.

Базовые блоки в каждой области посещаются в топологическом порядке. Такое упорядочение гарантирует, что базовый блок не будет планироваться до тех пор, пока не будут спланированы все команды, от которых он зависит.

87. Топологическая сортировка.

каждая вершина должна быть обработана до тех вершин, на которые она указывает

88. Программная конвейеризация.

Циклы, итерации которых независимы одна от другой называются универсальными.

Метод планирования кода, который позволяет в результате планирования процедур, содержащих универсальные циклы, получать достаточно компактный код, который будет как можно более полно использовать возможности компьютера по параллельному выполнению команд, или (для компьютеров классов VLIW и EPIC) операций (частей команд).

89. Ресурсы. Виды ресурсов.

Вектора R = [r1, r2, ...], представляющего аппаратные ресурсы, где ri - количество доступных единиц i-го ресурса

90. Таблица резервирования ресурсов.

Использование ресурсов для каждой машинной команды типа T моделируется двумерной таблицей резервирования ресурсов (Resource Reservation Table) RTi.

Ширина таблицы равна количеству видов ресурсов машины, а длина - продолжительности использования ресурсов в операции.

91. Пространственная локальность данных.

обращение к ячейке памяти означает, что через небольшой промежуток времени будет обращение к ячейкам, находящимся вблизи этой ячейки.

92. Временная локальность данных.

обращение к ячейке памяти означает, что через небольшой промежуток времени к этой ячейке снова будет обращение.

93. Для чего необходимо обеспечивать локальность данных?

повышает эффективность - чем ближе расположены, тем выше скорость

94. Распространение копий.

Распространение копий означает замену всех последующих вхождений переменной x на переменную y.

95. Распространение констант.

это оптимизирующее преобразование,
заменяющее выражения, которые при выполнении
всякий раз вычисляют одну и ту же константу,
самой этой константой.