Linear Scan Register Allocation

Распределение регистров

- назначение переменных и временных значений физическим регистрам процессора
- минимизация трафика между оперативной памятью и процессором
- количество регистров ограничено, поэтому важно их эффективное использование

Обращение к памяти

- переливание (spilling) - сохранение в стеке значения, которое не может быть сохранено в регистре, до того, как регистр будет перезаписан другим значением
- когда выгруженное значение используется позже, оно должно быть снова загружено в регистр из памяти

Классификация

- локальные методы ограничиваются небольшой частью скомпилированного в данный момент метода; идентифицируются и оптимизируются внутренние циклы
- глобальные методы пытаются оптимизировать целые методы или даже группы методов; эффективны, но медленнее

NP-полнота проблемы

- оптимальное глобальное и локальное распределение регистров - NP-полная проблема
- каждый алгоритм должен найти компромисс между временем компиляции при распределении и временем выполнения результирующего кода
- важно для JIT, где время компиляции является частью общего времени работы приложения
- в JIT чаще используются локальные методы

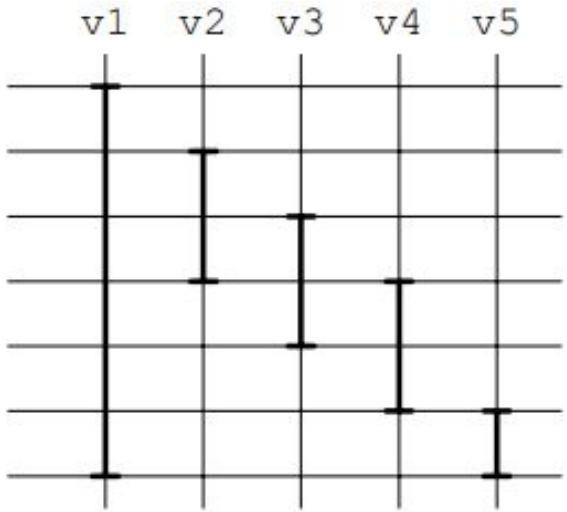
Алгоритм линейного сканирования

- базовая версия без учета дыр в интервалах времени жизни
- улучшенная версия (second chance binpacking) с учетом дыр в интервалах времени жизни и их разбиением на части во время распределения

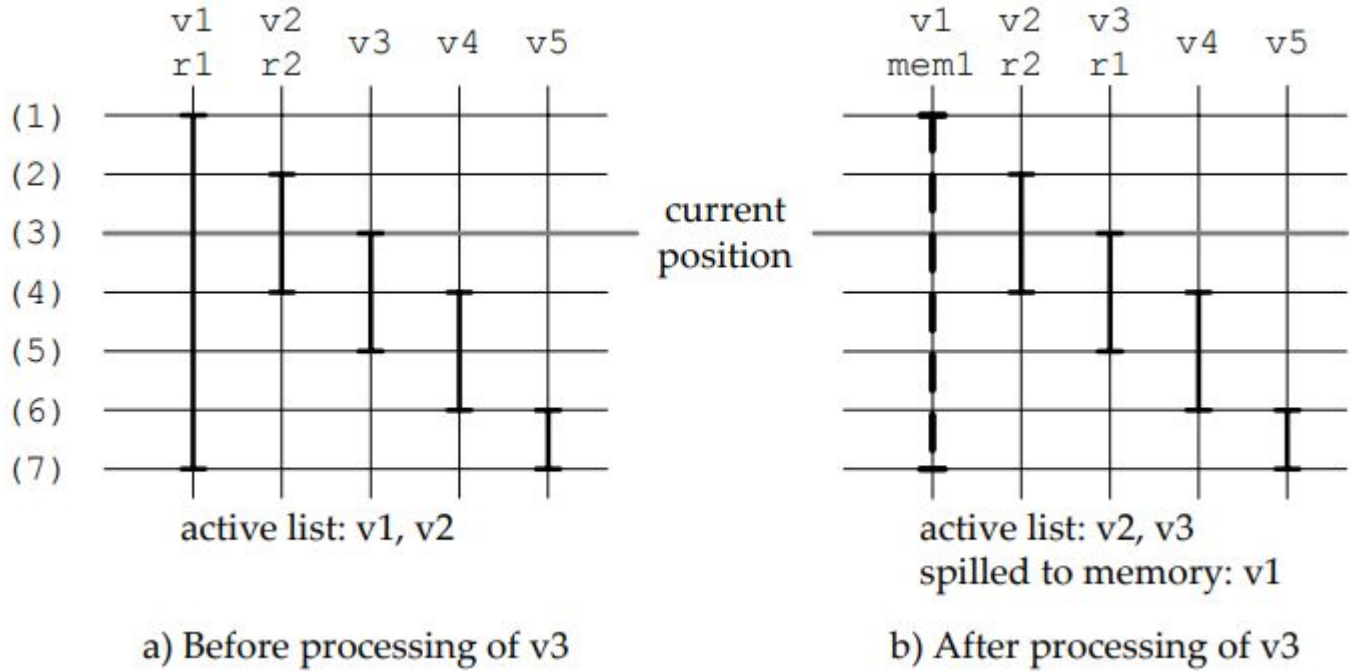
Базовый алгоритм

- доступные физические регистры - r1, r2
- активный список - все интервалы, перекрывающиеся с текущей позицией и имеющие уже назначенные регистры

```
(1)   v1 = 10
(2)   v2 = 20
(3)   v3 = v1 + v2
(4)   v4 = v2 + v3
(5)   v1 = v3 + v4
(6)   v5 = v4 + v1
(7)   return v1 + v5
```



Базовый алгоритм



Базовый алгоритм

```
(1)   v1 = 10           mem1 = 10
(2)   v2 = 20           r2 = 20
(3)   v3 = v1 + v2      r1 = mem1 + r2
(4)   v4 = v2 + v3      r2 = r2 + r1
(5)   v1 = v3 + v4      mem1 = r1 + r2
(6)   v5 = v4 + v1      r1 = r2 + mem1
(7)   return v1 + v5    return mem1 + r1
```

Базовый алгоритм: оценка

- \- значения некоторых виртуальных регистров должны быть отправлены в стек
- \+ это компенсируется более быстрым временем распределения

- \+ требуется только один линейный проход по интервалам времени жизни
- \+ асимптотическая временная сложность O(n), где n - количество виртуальных регистров

- ! компромисс, если важно время компиляции

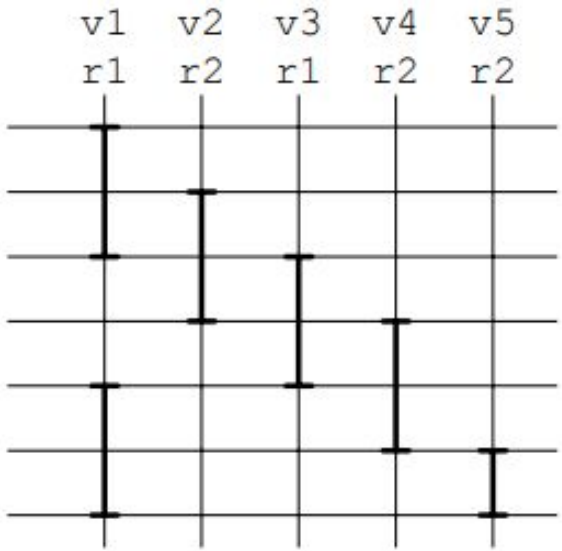
Улучшенный алгоритм (second chance binpacking)

+ способен обрабатывать дыры в интервалах времени жизни
+ способен разделять интервалы времени жизни
+ анализирует поток данных, чтобы минимизировать количество вставляемых перемещений

Улучшенный алгоритм

+ неактивный список - все интервалы, которые начинаются до и заканчиваются после текущей позиции, но в данный момент находятся в дырке времени

```
(1)   v1 = 10
(2)   v2 = 20
(3)   v3 = v1 + v2
(4)   v4 = v2 + v3
(5)   v1 = v3 + v4
(6)   v5 = v4 + v1
(7)   return v1 + v5
```




```
r1 = 10
r2 = 20
r1 = r1 + r2
r2 = r2 + r1
r1 = r1 + r2
r2 = r2 + r1
return r1 + r2
```

Улучшенный алгоритм: оценка

- \- общая асимптотическая временная сложность превышает O(n)
- \+ всего несколько процентов от общего времени распределения тратится на нелинейные части, поэтому жертвование линейностью не имеет большого влияния

! компромисс, если важны как время компиляции, так и время выполнения программы

Литература

- Linear Scan Register Allocation for the Java HotSpot™ Client Compiler, Christian Wimmer (2004)