Задача глобального распределения регистров во время динамической двоичной трансляции.

Кирилл Батузов

ИСП РАН

2 декабря 2016 г.

Распределение регистров

- Локальное распределение регистров.
  - Эвристические алгоритмы.
- Глобальное распределение регистров.
  - Метод раскраски графа взаимодействия переменных.
  - Метод линейного сканирования.

Глобальное распределение регистров

- Требуется разработать алгоритм глобального распределения регистров, ориентированный на динамическую двоичную трансляцию.
- Тестирование будет производиться в эмуляторе QEMU.
  - Распределение регистров совмещено с генерацией машинного кода.
  - Минималистичное внутреннее представление.

Основные идеи алгоритма

- Распределить глобальные переменные на регистры на границах базовых блоков непротиворечивым образом.
- Провести распределение регистров внутри базовых блоков модифицированным алгоритмом локального распределения регистров, который соблюдает граничные условия.

Обеспечение корректности граничных условий

Определение

Назовем условия на распределение регистров в начале базового начальными, а в конце — конечными. Введем следующие обозначения.

- Если есть базовый блок b, то начальные условия этого базового блока обозначим как b^{pre}, а конечные — как b^{post}.

Определение

Множество граничных условий блока трансляции TB с графом потока управления $G = \langle B, E \rangle$ назовем корректным, если

$$\forall (b_1, b_2) \in E : b_1^{post} = b_2^{pre}.$$

Обеспечение корректности граничных условий

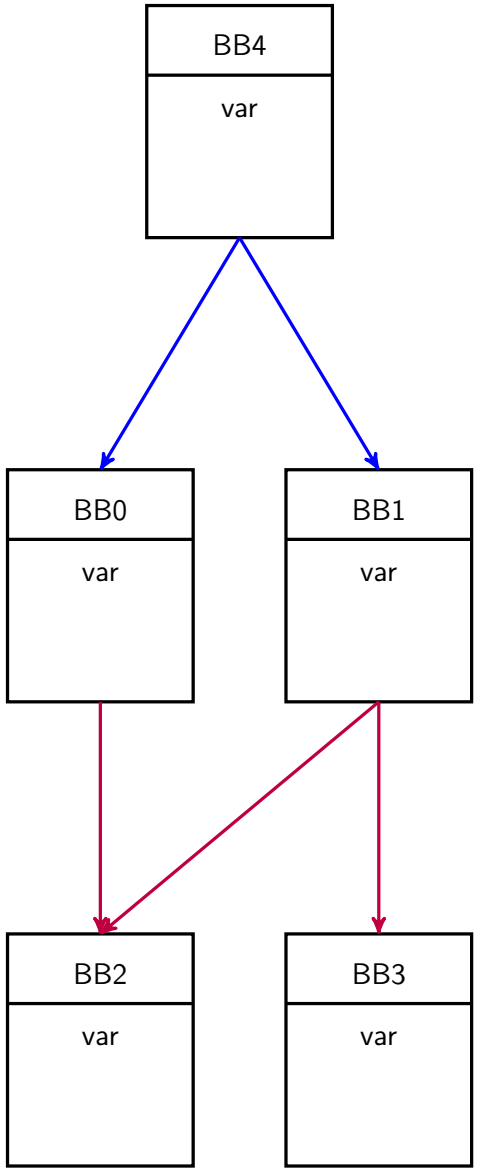
Определение

Дуги e_1 и e_2 графа потока управления назовем родственными, если они выходят из одного и того же блока либо входят в один и тот же блок. Обозначим $e_1 \sim e_2$.

Определение

Точкой синхронизации назовем непустое множество J дуг графа потока управления такое, что

- для любых двух родственных дуг e_1 и e_2 выполнено соотношение
$$e_1 \in J \iff e_2 \in J,$$
- для любых двух дуг u и v графа потока управления входящих в одну точку синхронизации существует последовательность дуг $e_1, e_2, \ldots, e_k$ таких, что $e_1 = u$, $e_k = v$, $\forall i \in [1, k-1]$ $e_i \sim e_{i+1}$.


BB4
var
BB0
var
BB1
var
BB2
var
BB3
var

Обеспечение корректности граничных условий

Утверждение

Пусть дуги $e_1 = (u_1, v_1)$ *и* $e_2 = (u_2, v_2)$ *графа потока управления принадлежат одной и той же точке синхронизации* J*. Тогда для любого корректного множества граничных условий выполнены равенства* $u_1^{post} = u_2^{post}$ *и* $v_1^{pre} = v_2^{pre}$*.*

Пусть дан блок трансляции TB с графом потока управления $G = \langle B, E \rangle$. Построим неориентированный граф $G_E = \langle E, F \rangle$, в котором $(e_1, e_2) \in F$ тогда и только тогда, когда $e_1 \sim e_2$.

Утверждение

Множество дуг $\{e_i\}$ *графа потока управления* $G = \langle B, E \rangle$ *является точкой синхронизации тогда и только тогда, когда они образуют компоненту связности в графе* G_E*.*

Комбинированный алгоритм

procedure COMBINED-REG-ALLOC(TB)
  for all $b \in B(TB)$ **do** ▷ Начальная инициализация
    $b^{pre} \leftarrow \varnothing$
    $b^{post} \leftarrow \varnothing$
  end for
  for all $J \in \mathbb{J}(TB)$ **do** ▷ Основной цикл
    $regmap \leftarrow$ COMPUTE-REGISTER-MAPPING(J)
    for all $(src, dst) \in J$ **do**
      $src^{post} \leftarrow regmap$
      $dst^{pre} \leftarrow regmap$
    end for
  end for
  for all $b \in B(TB)$ **do**
    LOCAL-REG-ALLOC(b)
  end for
end procedure

Выбор переменных для граничных условий

Введем функцию полезности включения переменной x в граничные условия точки синхронизации J: $\textsc{Usefulness}(x, J)$.

$$|\{(b_1, b_2) : (b_1, b_2) \in J \wedge x \in \mathit{Vars}(b_1) \wedge x \in \mathit{Vars}(b_2)\}|$$

Пусть выполнено одно из следующих условий:

$$|b^{\mathit{pre}}| + \textsc{Registers-Needed}(b) \leqslant \textsc{Total-Registers},$$

$$|b^{\mathit{post}}| + \textsc{Registers-Needed}(b) \leqslant \textsc{Total-Registers}.$$

Тогда множества регистров $\mathit{Regs}(b^{\mathit{pre}})$ или $\mathit{Regs}(b^{\mathit{post}})$ и множество регистров, используемых внутри базового блока, можно выбрать не пересекающимися.

Определение

Регистровым давлением в инструкции I из базового блока b называется минимальное количество регистров, необходимых для генерации кода данной инструкции в предположении, что все переменные, которые живы в данной точке базового блока и используются в нем в инструкции I или после нее, располагаются на регистрах.

$$\textsc{Reg-Pressure}(I, b) = |\textsc{Live-Variables}(I, b)| + |\textsc{Extra-Registers}(I)|$$

Определение

Регистровым давлением в базовом блоке b назовем максимальное среди регистровых давлений во всех его инструкциях.

$$\textsc{Reg-Pressure}(b) = \max_{I \in b} (\textsc{Reg-Pressure}(I, b))$$

Выбор количества регистров

Если в точке синхронизации J использовать не более

$$\textsc{Total-Regs} - \max_{(b_1,b_2)\in J} (\textsc{Reg-Pressure}(b_1)), \textsc{Reg-Pressure}(b_2))$$

регистров для граничных условий, то соотношения

$$|b^{pre}| + \textsc{Reg-Pressure}(b) \leqslant \textsc{Total-Registers}$$

$$\left|b^{post}\right| + \textsc{Reg-Pressure}(b) \leqslant \textsc{Total-Registers}$$

будут выполнены.

Алгоритм выбора граничных условий

function COMPUTE-REGISTER-MAPPING(J)

$psrc \leftarrow \max\limits_{(b_1,b_2)\in J}(\text{REG-PRESSURE}(b_1))$

$pdst \leftarrow \max\limits_{(b_1,b_2)\in J}(\text{REG-PRESSURE}(b_2))$

$p \leftarrow \max(psrc, pdst)$

$n \leftarrow \text{TOTAL-REGS} - p$

$result \leftarrow \varnothing$

$priority \leftarrow \text{COMPUTE-USEFULNESS}(J)$

for $i \leftarrow 1, n$ **do**

$result \leftarrow result \cup \{(priority[i], register[i])\}$

end for

return *result*

end function

Изменения алгоритма локального распределения регистров

- Начальная инициализация должна происходить в соответствии с предусловиями.
- В конце базового блока необходимо привести распределение регистров в соответствие с постусловиями.

Алгоритм, решающий последнюю задачу, назовем алгоритмом переупорядочивания регистров.

Алгоритм переупорядочивания регистров

Обозначим текущее распределение регистров b^{cur}. Построим ориентированный граф G_r, вершинами которого будут являться регистры целевой архитектуры. Дуга (r_1, r_2) присутствует в графе тогда и только тогда, когда существует переменная v такая, что $(v, r_1) \in b^{cur}$, $(v, r_2) \in b^{post}$. То есть содержимое регистра r_1 необходимо переместить в регистр r_2.

По определению граничных условий в графе G_r в каждую вершину входит не более одной дуги и выходит также не более одной дуги. Значит, граф представляет собой совокупность цепей, циклов длины больше единицы и петель.

Алгоритм переупорядочивания регистров

- Операция $\textsc{Spill}$ может быть применена только к регистру, в котором хранится некоторая переменная. При этом дуга, выходящая из соответствующей вершины исчезает если она была.
- Операция $\textsc{Load}$ может быть применена только к регистру, в котором не хранится никакая переменная. При этом появится дуга, выходящая из соответствующей вершины. Случай загрузки переменной, не входящей в множество $Var(b^{post})$, рассматриваться не будет.
- Операция $\textsc{Move}$ может быть применена только к паре регистров (r_1, r_2) таких, что в r_1 не хранится никакая переменная, а в r_2 хранится некоторая переменная. При этом если из вершины r_2 выходила дуга $e = (r_2, r)$, то она исчезнет, а вместо нее добавится дуга $e' = (r_1, r)$.

Алгоритм переупорядочивания регистров

1. Все регистры, содержащие переменные, не входящие во множество $Vars(b^{post})$, освобождаются с помощью операций $\textsc{Spill}$. После этого шага все регистры, из которых в графе G_r не выходит дуги являются свободными.
2. До тех пор, пока существует пара регистров (r_1, r_2), такая что в G_r есть дуга из r_2 в r_1 и нет дуги исходящей из r_1, к ним применяется операция $\textsc{Move}(r_1, r_2)$. В результате это операции исчезает дуга (r_2, r_1). После завершения данного шага в графе G_r не останется цепей.

Алгоритм переупорядочивания регистров

3. До тех пор, пока в графе G_r существует цикл длины больше единицы, он «разрывается», а шаг 2 повторяется. «Разорвать» цикл можно двумя способами:
   - переместив с помощью операции $\textsc{Move}$ содержимое одного из регистров, входящих в цикл, в свободный;
   - сбросив содержимое одного из регистров, входящих в цикл, в память, с помощью операции $\textsc{Spill}$.
4. После завершения предыдущего шага в графе G_r остались только петли. Осталось загрузить на регистры недостающие переменные (то есть переменные из множества $Vars(b^{post}) \setminus Vars(b^{cur})$). Все нужные регистры уже свободны.

Определение

Пусть есть два алгоритма переупорядочивания регистров. Алгоритм A_1 генерирует L_1 операций $\textsc{Load}$, S_1 операций $\textsc{Spill}$ и M_1 операций $\textsc{Move}$. Алгоритм A_2 генерирует L_2 операций $\textsc{Load}$, S_2 операций $\textsc{Spill}$ и M_2 операций $\textsc{Move}$. Алгоритм A_1 эффективнее алгоритма A_2 тогда и только тогда, когда $L_1 + S_1 < L_2 + S_2$ либо $L_1 + S_1 = L_2 + S_2$ и $M_1 < M_2$.

Определение

Алгоритм A переупорядочивания регистров является оптимальным, если не существует алгоритма A' эффективнее него.

Утверждение

Приведенный алгоритм является оптимальным среди алгоритмов переупорядочивания регистров.

Экспериментальные результаты

- Ускорение на искусственном примере — 29%.
- На реальных примерах — ускорение компенсируется дополнительными накладными расходами.
- Очень мало блоков трансляции, к которым алгоритм применим — около 10%.

Дальнейшие работы

- Увеличение размера блков трансляции.
- Улучшение взаимодействия локального и глобального распределения регистров.

Спасибо за внимание!