Static Links and Register Allocation

Sourabh Aggarwal

1 November 2019

Department of Computer Science And Engineering
IIT Palakkad

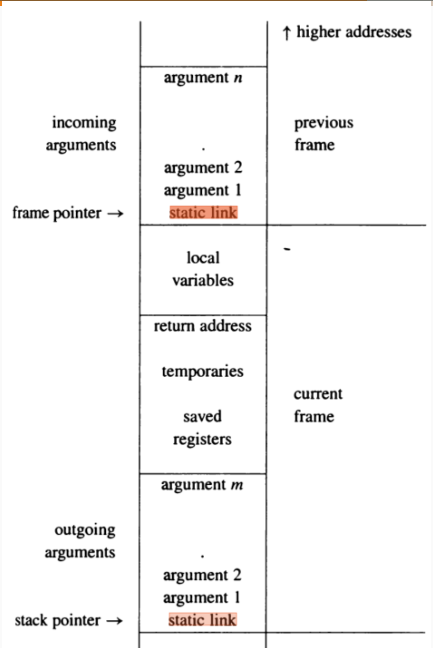
Escaping of Variables

- Some variables must go into memory and not in register.

Escaping of Variables

- Some variables must go into memory and not in register.
- If a variable is accessed in a nested function then that variable escapes.

Static Links



Code demonstration

Liveness Analysis

- Two temporaries `a` and `b` can fit into the same register, if `a` and `b` are never ”in use” at the same time.
- We say a variable is live if it holds a value that may be needed in the future, so this analysis is called liveness analysis. To perform analyses on a program, we make a control-flow graph. Each statement in the program is a node in the flow graph; if statement `x` can be followed by statement `y` there is an edge from `x` to `y`.
- Liveness of node is calculated as shown:
  $in[n] = use[n] \cup (out[n] \setminus def[n])$
  $out[n] = \cup_{s \in succ[n]} in[s]$

Liveness Analysis

- A condition that prevents `a` and `b` being allocated to the same register is called an interference.
- The most common kind of interference is caused by overlapping live ranges when `a` and `b` are both live at the same program point, then they cannot be put in the same register.
- Interference graph is created with vertices as temporaries and edges as follows:-
    - At any nonmove instruction that defines a variable a, where the live-out variables are $b_1, \dots, b_j$, add interference edges $(a, b_1), \dots, (a, b_j)$.
    - At a move instruction $a \leftarrow c$, where variables $b_1, \dots, b_j$ are live-out, add interference edges $(a, b_1), \dots, (a, b_j)$ for any b_i that is not the same as c.

Register Allocation

- Now that we have made our interference graph, where each edge denotes that its endpoints must go in different registers, thus, our problem reduced into coloring our graph with K (# no. of available registers) colors such that no two end points have same colour.

Register Allocation

Recursive algorithm is defined as below, assume that we are given a set of instructions each of which may contain temporaries which might have been already mapped to some register.

- Construct interference graph given the list of instructions.
- Start with vertices (temporaries) which have not been assigned a colour (register).
- Such vertices whose # Neighbours < K can be removed from the graph as we would always have a color available for them. Thus this divides our list of vertices into two viz., (simplify, potentialSpill).
- Now we should process these nodes in simplify, let "stack" denote the set of simplified nodes.
- Repeat until potentialSpill nodes become empty
    - We "simplify" each node in our simplify list by removing it from simplify list and adding it to stack and decrementing the degree of its neighbors which are not in stack (stack nodes are already removed) and also which aren't already colored. Note that when we decrement the degree of the neighbors, if degree becomes less then K that means we can add this node to simplify and remove it from potentialSpill.
    - If potentialSpill is empty then done, o/w select a node to spill, choose that which has maximum neighbors and should have accessed temps/memory least number of times. Assign it to simplify and remove it from potentialSpill.
- Now process stack from top to bottom. For an element, colors available to it are total minus those taken by its neighbors, if colors are available then assign else we add it to actualSpill.
- If by now actualSpill is empty then we stop else we rewrite the program and repeat.

Thanks!