Lecture 20: Implementing local register allocation

David Hovemeyer

November 15, 2023

601.428/628 Compilers and Interpreters

Today

- Implementing register allocation

Implementing local register allocation

Local register allocation

Goal of register allocator is to assign (temporarily!) machine registers to virtual registers

- Special case: virtual registers that are alive at the end of the basic block shouldn't have a temporary register assignment
- Could leave these allocated in memory, or use "long term" register assignment (i.e., callee-save registers)

Problem: there are a limited number of machine registers

- If we run out, steal a machine register that is currently in use, first spilling its value to memory
- Bottom-up register allocation: when stealing, choose the virtual register whose next def is the furthest in the future

Register allocator state

Information to keep track of as allocator progresses through instructions in basic block:

- Collection of available machine registers (stack or queue)
- Map of virtual register numbers to assigned machine register
- Collection of available spill locations (stack or queue)
- Map of virtual register numbers to spill locations

Recording register assignments

The register allocator will need to communicate register assignments to the low-level code generator.

One way to do this: add a field to Operand, if set to a non-negative value, it's the assigned machine register.

Making allocations and assignments

For each virtual register used in an instruction[1]:

- If its value is currently spilled, allocate a machine register and restore it
- If there is a current assignment to a machine register, record the assignment
- If there is no assignment, allocate a register and record the assignment

[1] Except for vregs excluded from being assigned a temporary register.

Allocating a register

- If a machine register is available, allocate it (easy case)
- If no machine register is available (harder case):

1. Choose a victim vreg
2. Allocate a currently-unused spill location, otherwise, use a new spill location
3. Emit a spill instruction (specifying the vreg, mreg, and spill location)
4. Use the stolen mreg to satisfy the allocation

Restoring a spilled register

Assuming that an machine register has already been allocated:

1. Emit a restore instruction (specifying vreg, mreg, and spill location)
2. Return the (no longer used) spill location to the collection of available spill locations

Allocating storage for spill locations

Determine maximum number of spill locations used (over all basic blocks)

Place storage area for spills somewhere in the stack frame

Low-level code generator will need to determine an offset into the storage area for each spill and restore

Dealing with procedure calls

The compiler must assume that a call to a procedure could change the value of any caller-save register! (e.g., %rcx, %rdx, %r10, etc.)

Should not be a huge problem in practice:

- If a basic block has a `call` instruction, it will be the last instruction
- Local register allocation should only assign machine registers to vregs used for temp values: these values will be dead at the end of the basic block[2]
- The register allocator should avoid allocating machine registers that will be needed to pass arguments

[2] In general, don't allocate registers to vregs that aren't dead at the end of the block