Lecture 22: Code optimization strategy

David Hovemeyer

November 28, 2022

601.428/628 Compilers and Interpreters

Today

- Expectations for code optimization (Assignment 5)
- Possible approach

Expectations for Assignment 5

- Assignment 5 is open-ended
- No inherently right way to approach it
- Primary expectations:
  - Identify opportunities to improve generated code
  - Implement optimizations to address those opportunities
- It will be most straightforward to focus on optimizing the high-level code

Focus on local optimizations

- To avoid complexities arising from control flow, we recommend implementing *local* (basic-block scope) optimizations
- The `ControlFlowGraphTransform` class automates transformation of a CFG
    - Override the `transform_basic_block` member function
- Use `HighLevelControlFlowGraphBuilder` to build a high-level `ControlFlowGraph` object from the high-level `InstructionSequence`

Minimum expectation

- At a minimum, you should do *something* to improve the generated code
- Some relatively easy optimizations
  - Constant folding
  - Constant propagation
  - Copy propagation
  - Dead store elimination

Peephole optimizations

- A *peephole* optimization scans a basic block to look for short sequences of consecutive instructions which have some obvious and easy-to-fix inefficiency
- Useful as a way to “clean up” the generated code
- These can be quite effective, and can be relatively easy to implement
- Could be useful on both high- and low-level code

Local value numbering

- Local value numbering (if implemented fully) subsumes constant folding and constant propagation, and also eliminates redundant computations
- But, it's fairly challenging to implement!
- It is definitely not *mandatory* to implement this

Local register allocation

- Local register allocation is relatively straightforward to do, and should get you a considerable speedup
- Idea is to scan high-level instructions in each basic block, and assign machine registers to virtual registers instruction by instruction
  - Personally, I find bottom-up register allocation to be the most intuitive approach
- You'll need to allocate memory in the stack frame for spilled registers
- *Important*: do not assign machine registers to any virtual registers which are live at the end of the basic block

“Global” allocation of callee-saved registers to local variables

- Local register allocation can’t allocate registers for local variables whose lifetimes are greater than one basic block
- *However*, you could do a *global* (entire function scope) allocation of a callee-saved register to a local variable
- Idea: identify “frequently-used” variables (e.g., loop variables), and “pre-allocate” callee-saved registers to them
  - These allocations are in effect for the entire function (hence, they are “global”)
- The local register allocator will need to be aware of such assignments
- This is a very easy way to allocate registers for loop variables

Scenario 1

- Do some basic local optimizations (constant folding, constant propagation, copy propagation, dead store elimination)
- Do some peephole optimizations?
- If done well (and with good experiments and report) this could reach the B to B+ range

Scenario 2

- Some basic local optimizations, plus local register allocation
- Maybe allocation of callee-saved registers to loop variables
- If done well (with good experiments and report), this could reach the A-range

Scenario 3

- Local value numbering (with associated “cleanup” passes, such as copy propagation and dead store elimination)
- Local register allocation
- Maybe allocation of callee-saved registers to loop variables
- If done well (plus good experiments/report), should be a solid A

If you are feeling ambitious

Some ideas for “above and beyond” level code optimization:

- Implement a dataflow analysis (see me if you are interested in trying this, there is a general framework for dataflow analysis in the starter code)
- “Advanced” instruction selection techniques (perhaps replace address computation with indexed or index/scaled addressing modes)
- Global register allocation (had one student do this successfully last year!)