15-411: Compiler Design — Spring 2024

Recitation 9: Optimizations II and Lab 6 — 29 March

Register Coalescing

Register coalescing is a useful optimization for removing moves between registers. It is based on the simple observation that if two different temps do not have an interference edge between them, then they could both be renamed to be the same temp without changing the meaning of the program.

The greedy algorithm proposed by Pereira and Palsberg occurs *after* we have colored the interference graph but *before* we have rewritten the program to replace temps with registers. It is as follows:

(1) Consider each move between variables $t \leftarrow s$ occurring in the program in turn.

(2) If there is an edge between t and s, that is, they interfere, they cannot be coalesced.

(3) Otherwise, if there is a color $c \leq c_{\text{max}}$ which is not used in the neighborhoods of t and s, i.e., $c \notin N(t) \cup N(s)$, then the variables t and s are coalesced into a single new variable u with color c:

   (a) Create a new node u with color c and create edges from u to all vertices in $N(t) \cup N(s)$.

   (b) Remove t and s from the graph.

   (c) Replace t and s with u in the program.

The color c_{max} is the maximal color that has been used in the coloring of the original graph. Because of the tested condition, the resulting graph is still K-colored, where K is the number of available registers. Of course, we also need to eventually rewrite the program appropriately by replacing both t and s with u everywhere so that the program remains in correspondence with the graph.

A Note on SSA

SSA is very useful to implement since many optimizations can be easily and correctly implemented over it, as mentioned before. However, SSA itself can be tricky to implement. These references may be helpful to read about SSA implementation intricacies in more detail:

(a) https://link.springer.com/book/10.1007/978-3-030-80515-9

(b) https://homepages.dcc.ufmg.br/ fernando/publications/papers/CC09.pdf

Aliasing

As discussed in lecture, memory aliasing can interfere with various optimizations such as constant propagation and common subexpression elimination. One clear way to circumvent this problem is to avoid propagating values assigned to memory locations entirely. On the other hand, sound alias analysis can help us determine whether or not a value at a memory location has changed–and can therefore be propagated forward.

Checkpoint 0

Consider all the variables in this function, which ones can be aliases of others? Which values are safe to propagate?

```
1 \\ bar is a library function
2 void bar (bool* a, int* b);
3
4 int foo () {
```

```
  int* x = alloc(int);
  int* y = alloc(int);
  *x = 1;
  *y = *x;
  int *z = x;
  bool* w = alloc(bool);
  bar(w, y);
  if (*w) return *y;
  return *y + *z;
}
```

Lab 6

Lab 6 involves extending your compiler with some interesting new feature(s). Each team is asked to choose one of the following options for their project:

(1) targeting LLVM bytecode, or

(2) extending the source language of your compiler to `C1` (and beyond), or

(3) implementing garbage collection, or

(4) defining and implementing your own project ("choose your own adventure").

For each of these options, you will generally be expected to submit the following items:

- A working compiler and support materials (runtime, etc.) that implement your project, and
- Any additional test cases and framework for testing your project, and
- A term paper describing and critically evaluating your project.

Choose Your Own Adventure Proposal

Before Lab 6 officially begins, we ask you to write a proposal for a possible "choose your own adventure" project. Choose-your-own-adventure ideas that have been pursued in the past include:

- New language features for C0 (functional programming constructs, exception handling, . . .)
- New compilation targets/backends
- Additional safety/security guarantees (via the type system or static analysis passes)
- Major, complex optimizations

You should complete this written description of a new, unique project even if you plan to implement one of the preset options for Lab 6. This is meant to get you thinking about ways to extend your compiler beyond the preset options for Lab 6. While we **will not** require you to implement your exact proposal for Lab 6, this is still a good opportunity to get feedback from the course staff about how reasonable your proposal is.

Have fun with this! If you have any questions about the assignment (or would like to talk out if something is a "reasonable" amount of work), please post on Piazza or talk to the course staff at office hours.