Project 1 Report

Joseph A. Boyle

March 4, 2018

Abstract

In this report, we explore different types of local register allocators and their effectiveness. Specifically, we test two versions of the Top-Down Allocator, each with their own selection heuristic, and a Bottom-Up Allocator. The results of these allocation techniques are evaluated on the number of cycles and instructions needed for execution, through a variety of samples of ILOC code.

1 Introduction

We are presented with several snippets of ILOC code which utilize an unbounded number of registers, and utilize one of three allocation techniques to limit the number of registers used to a set number, k, while still maintaining code semantics.

1.1 Bottom-Up Allocation

In the Bottom-Up allocation scheme, we maintain a list of k registers, and note which of those k are currently free. When a register is needed by an instruction, we can either use one of those free registers, or if none exist, spill one of the in-use registers into memory. In-use registers that are good candidates for spilling are those which aren't going to be used for the longest amount of time in the future, therein allowing other values to take its place, reducing the number of loads for closer instructions. Not depending on a feasible set, F, but rather using the given register for loading/storing, all k physical registers may be used for allocation.

1.2 Top-Down Allocation

Top-Down allocation performs a scan of the local block, sorting each virtual register by its number of uses. Using some selection heuristic, we decide which of the virtual registers are best to keep in physical registers and which should be loaded in as needed, generally sorted by usage. These type of allocators require some feasible set, F, of registers for loading/storing values, and as such we are only allowed to use $(k-F)$ registers for allocation, reserving the other F registers for loading/storing values from memory as needed.

There are two variants of Top-Down allocation tested in this report, one which was shown in class, utilizing live ranges, and one that is demonstrated in *Engineering: A Compiler*. Both sort by the number of occurrences of each virtual register, but the method shown in class uses the length of the liverange as a tie-breaker.