Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Eliminating Small, Transient Memory Allocations

Dário Tavares Antunes
B. A. (Mod.) Computer Science
Final Year Project May 2018
Supervisor: Dr. David Abrahamson

School of Computer Science and Statistics
O'Reilly Institute, Trinity College, Dublin 2, Ireland

Declaration

I hereby declare that this project is entirely my own work and that it has not been submitted as an exercise for a degree at this or any other university.

Dário Tavares Antunes, January 1, 1970

Permission To Lend

I agree that the Library and other agents of the College may lend or copy this report upon request.

Dário Tavares Antunes, January 1, 1970

Abstract

This is where I would put the abstract, IF I HAD ONE

Acknowledgements

This is where I would put my acknowledgements, IF I HAD ANY

Contents

Chapter 1

Introduction

> 640K ought to be enough for anybody.
>
> Not Bill Gates

Despite the often misattributed epigraph above often being used to mock past beliefs that some amount of memory should be enough for any reasonable purposes, the mentality behind it is still pervasive.

With the broad availability of large amounts of computational power, memory and storage, conservation or efficient use of the same is often overlooked in programming. This is largely perpetuated by the (often valid) view that programmer time is more valuable than the benefits that more efficient but more complex code brings.

However, there remain situations where these benefits are in fact worth the effort required. One of these such cases is in code intended to be deployed in embedded or mobile devices, where resources are limited and preservation of power is essential.

A blog post [Ste17] by Daniel Stenberg, original author of the `curl` command line tool and ubiquitous URL data transfer tool, is a retrospective on an attempt to reduce unnecessary heap allocations.

Inspired by that post, the aim of this project is to produce a tool to identify cases where similar changes could be made in order to potentially reduce a program's energy and processing power footprint, and at the same time improve its performance.

Chapter 2

Background and Objectives

This chapter will contain:

- background on the patch as applied by Daniel Stenberg
- hypothesis as to why the patch caused a performance increase (why that much vs more/less)
- background based on similarity to generational GC assumptions (Appel, Shao)
- predictions on the results of a generalised application of the patch (vs proebsting's law for compilers)

Chapter 3

Implementation

This chapter will contain descriptions of:

- the goals of the plugin
- the frama-c platform
- difficulties encountered developing the plugin (new language, installation, documentation, usefulness of results from EVA [no location data for example], any future issues encountered)
- the state of the plugin at the end of the project, along with indications to refer to the future work section

Chapter 4

Case Studies

This chapter will include, for each chosen case study:

- indication of why that particular program was chosen
- expected results of applying the optimisation
- either: inlined patch code and commitish to apply it to OR indication of where to obtain the patch code/patched version
- results of the benchmark
- comparison to individual prediction, general prediction
- hypothesis for results

Chapter 5

Conclusion

This chapter will include high level summaries and conclusions on:

- the results
- the state of the plugin
- the potential benefit (or not) of further work

Chapter 6

Future Work

> A program is never less than 90% complete, and never more than 95% complete.
>
> — Terry Baker

Any non-trivial work is never complete. To that end, listed below are some ideas for potential improvements on the *forgetful* plugin or related works on the same principle.

6.1 Detecting Arbitrary Memory Allocations

The current implementation only finds allocations based on uses of malloc and free. Other ways to allocate memory exist (`calloc`, `realloc`, `alloca`, direct uses of `mmap` and `sbrk`), and platforms that stand to gain the most from this optimisation may have their own implementations.

An extension to this work could involve allowing an arbitrary list of functions declared to allocate or deallocate memory, potentially with fully annotated files specifying their behaviour so that `frama-c` can be used to its full potential (particularly for value analysis, which relies on these specifications).

Alternatively, if there is a willingness to assume a unix-like platform, the depth of analysis could be extended to attempt to automatically determine which functions might allocate memory by searching for `mmap` or `sbrk` calls and propagating annotations indicating functions that directly or indirectly allocate memory.

The approach propagating allocation information already exists in some form in Facebook's Infer [Fac13] static analyser, so future work could also involve extending that platform instead.

6.2 Automatically Performing Fixes

Ideally, fixes would be automatically generated and patched into the code at compile time, avoiding added complexity from the programmer's point of view while still taking advantage of the performance and memory benefits.

Potential intermediate steps toward that goal could involve generation of patches that could be applied to code before compilation, introducing the optimisation. Fortunately, `frama-c` already has code generation capabilities which could be taken advantage of for this purpose.

Bibliography

[Fac13] Facebook. *Facebook Infer*. 2013. URL: http://fbinfer.com/ (visited on 03/01/2018).

[Ste17] Daniel Stenberg. *Fewer mallocs in curl*. 2017. URL: https://daniel.haxx.se/blog/2017/04/22/fewer-mallocs-in-curl/ (visited on 02/01/2018).