Predicting Function Merging Profitability with Deep Learning

Rodrigo Rocha [1] **Pavlos Petoumenos** [2] **Zheng Wang** [3] **Murray Cole** [1] **Hugh Leather** [1]

Abstract

1 Introduction

In recent years, the market for mobile and embedded systems has been rapidly growing. These systems must often run on inexpensive and resource-constrained devices, with limited memory, storage, CPU caches. Applications for these systems are designed and built with different goals compared to traditional computing systems since their binaries must fit in a budget memory size. Hence, compilation techniques must be primarily focused on optimising for binary size.

One important optimisation capable of reducing code size is function merging. In its simplest form, function merging reduces replicated code by combining multiple identical functions into a single one (llv; Liška, 2014). More advanced approaches can identify similar, but not necessarily identical, functions and replace them with a single function that combines the functionality of the original functions while eliminating redundant code. At a high level, the way this works is that code specific to only one input function is added to the merged function but made conditional to a function identifier, while code found in both input functions is added only once and executed regardless of the function identifier.

Recent work has generalized function merging to any arbitrary pair of functions. The state-of-the-art (Rocha et al., 2019; 2020) first represents functions as nothing more than linear sequences of instructions and labels. Then it applies a sequence alignment algorithm, developed for bioinformatics, to discover the optimal way to create pairs of mergeable instructions from the two input sequences. Finally, it generates the merged function where aligned pairs of matching instructions are merged to a single instruction, while non-matching instructions are simply copied into the merged function.

[*]Equal contribution [1]School of Informatics, University of Edinburgh, United Kingdom [2]University of Manchester, United Kingdom [3]University of Leeds, United Kingdom. Correspondence to: Cieua Vvvvv <c.vvvvv@googol.com>, Eee Pppp <ep@eden.co.uk>.

Proceedings of the 3^{rd} MLSys Conference, Austin, TX, USA, 2020. Copyright 2020 by the author(s).

The state of the art optimization has a search strategy based on ranking the function candidates with higher similarity. However, because this strategy is unable to decide which one of those pairs are actually worth merging, most of the functions are actually unprofitable candidates. As a result, most of the merged function produced are thrown away and the original functions kept. Even if we consider only the top-ranked candidate functions, about 82% of the top ranked candidate functions are actually unprofitably merged.

Since the function merging operation is computationally expensive, we should avoid wasting time with unprofitable merged functions, freeing the compiler to focus more its efforts optimizing code that more likely to be profitable.

In this paper, we describe our heuristic model based on deep-learning that predicts whether or not a pair of functions can be profitably merged, avoiding wasteful merge operations.

2 Background

3 Motivation

In this section, we discuss two weaknesses in the state-of-the-art function merging optimization (Rocha et al., 2020). First, we show that there is a significant opportunity in reduction that can be gained by having a better profitability analysis. Second, we show that most function merging attempts are trown away as they cause code bloat, according to the profitability analysis.

3.1 Inaccuracies in the Profitability Analysis

Although the proposed technique is able to merge any two functions, it is not always profitable to do so. A pair of functions can be profitably merged when replacing them by the merged function results in an overall smaller code. As it is only profitable to merge functions that are sufficiently similar, for most pairs of functions, merging them increases code size. Since the profitability analysis is critical for the optimisation strategy, we must be able to effectively decide which pair of functions can be profitably merged.

In order to estimate the code-size benefit, we first estimate the size of all three functions, i.e., the two input functions and the merged one. The size of each function is estimated by summing up the estimated binary size of all instruction in the function, in its IR form. The binary size of each IR instruction is estimated by querying the compiler's built-in target-specific cost model. These cost models provide target-dependent cost estimations approximating the code-size cost of an IR instruction when lowered to machine instructions.

As pointed out by many prior work (**?**Rocha et al., 2019; 2020), even though cost models offer a good trade-off between compilation time and accuracy, they are expected to contain inaccuracies. Because we are trying to estimate the binary size of the final object code, inaccuracies arise from the fact that we are operating on the IR level and one IR instruction does not necessarily translate to one machine instruction. Because we are operating on the IR level, We cannot know exactly how each IR instruction will be lowered without actually running the compiler's backend. Moreover, several number of optimisations and code transformations will still run prior to machine code generation.

Figure 1 presents the code size reduction that can be achieved with an oracle. This oracle measures code size by compiling the whole program down to its final object file, which provides perfect information for the cost model. It shows the potential for improvement there exists from having a better profitability analysis, almost doubling the reduction. By compiling the whole program down to its binary form, we are able to precisely capture the impact on other optimizations and machine code generation, as well as the overheads that result from replacing the callsites to the merged function or keeping a *thunk*.

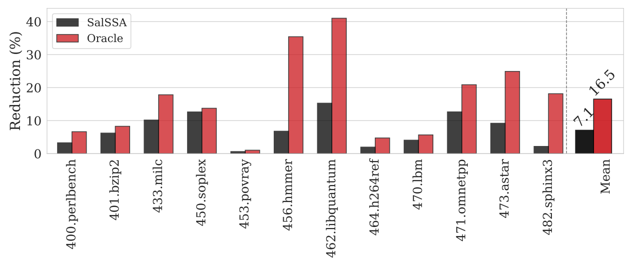

Figure 1. An average of about 82% of merging attempts are unprofitable.

However, the cost of compiling the whole program for every merging attempt is prohibitive. The re-compilation overhead can be severely aggravated for larger programs with multiple functions, where not only each compilation takes longer but the whole program is also re-compiled many times.

3.2 Wasteful Merge Operations

The fingerprint-based ranking strategy helps the function merging optimization to pair functions that are more similar. However, the current strategy is unable to decide which one of those pairs are actually worth merging. Figure 2 shows that about 82% of the top ranked candidate functions are actually unprofitably merged. As a result, a considerable amount of compilation time is wasted producing merged functions that will be thrown away, keeping the original pair of functions.

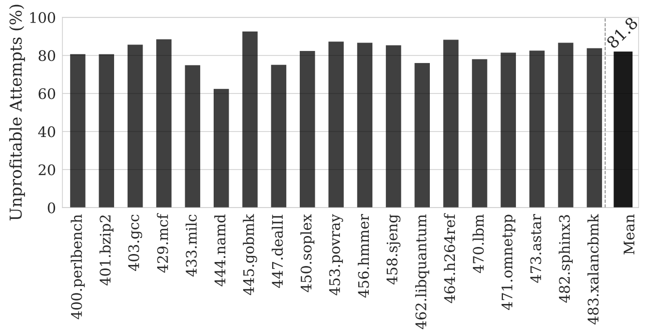

Figure 2. An average of about 82% of merging attempts are unprofitable.

Figure 3 shows the time spent producing unprofitable merged functions relative to the total compilation time of the function merging optimization. Since most of the merged functions are thrown away for being unprofitable, it is expected that most of the compilation time is also spent producing those merged functions. However, this impact is also aggravated when several of the unprofitable merged functions are much larger than the profitable ones.

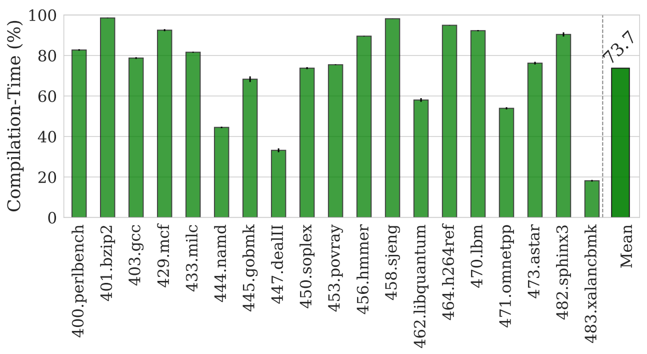

Figure 3. The percentage of compilation-time spent on unprofitable merge operations.

Therefore, it is of utmost importance that we avoid merging unprofitable functions. If we could eliminate all the time wasted on unprofitable merge operations, we would free compilation time for more useful computation.

3.3 Summary

In this paper, our goal is to develop a solution capable of identifying whether or not a given pair of functions can be profitably merged, allowing us to use a more expensive profitability analysis based on partial re-compilation. If we could predict which pairs of function are more likely to cause code bloat, we could avoid wasting time merging them in the first place and having to estimate their binary sizes. Bailing out early frees time to be spent on more profitable merge operations.

4 Our Approach

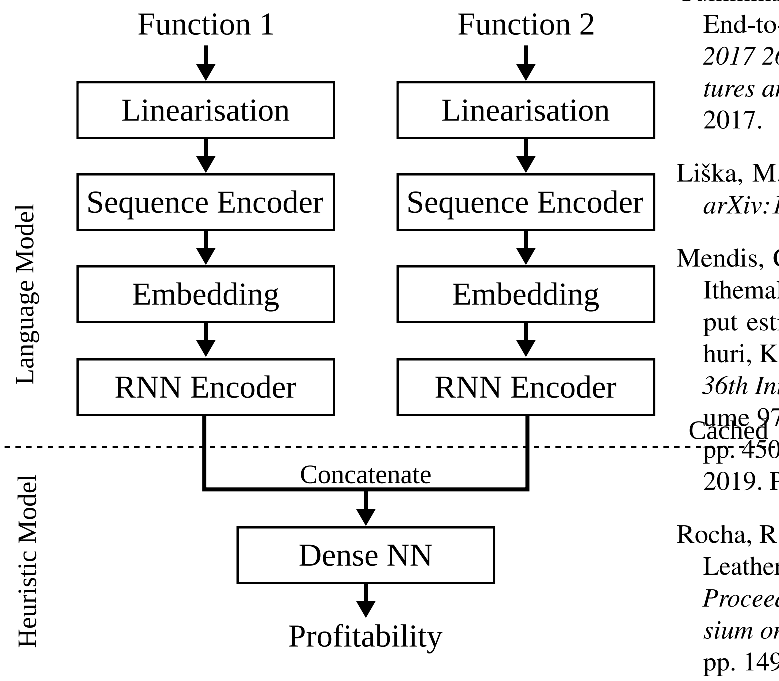

Figure 4. The proposed deep-learning model architecture that predicts which pairs of functions will be profitably merged. Code properties are extracted from each function into *context vectors* by the language model. These context vectors are cached to be later fed to the heuristic model to produce the final profitability prediction.

Figure 4 provides an overview of the prediction mechanism. Our mechanism follows a similar approach to previous deep-learning techniques for tuning compilers (Cummins et al., 2017; Mendis et al., 2019).

The same linearised functions used for the merge operations, as described in Chapter **??**, are used as input to the prediction model. First, we use a language model based on recurrent neural networks to encode the input functions into context vectors of fixed size. These vector encodings can be computed only once per function an cached. Finally, the context vectors of two input functions are concatenated and fed to a dense deep neural network to classify whether or not those functions are profitably merged.

5 Evaluation

6 Related Work

7 Conclusion

References

The LLVM Compiler Infrastructure. MergeFunctions pass, how it works. http://llvm.org/docs/MergeFunctions.html.

Cummins, C., Petoumenos, P., Wang, Z., and Leather, H. End-to-end deep learning of optimization heuristics. In *2017 26th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pp. 219–232, 2017.

Liška, M. Optimizing large applications. *arXiv preprint arXiv:1403.6997*, 2014.

Mendis, C., Renda, A., Amarasinghe, D., and Carbin, M. Ithemal: Accurate, portable and fast basic block throughput estimation using deep neural networks. In Chaudhuri, K. and Salakhutdinov, R. (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 4505–4515, Long Beach, California, USA, 09–15 Jun 2019. PMLR.

Rocha, R. C. O., Petoumenos, P., Wang, Z., Cole, M., and Leather, H. Function merging by sequence alignment. In *Proceedings of the 2019 IEEE/ACM International Symposium on Code Generation and Optimization*, CGO 2019, pp. 149–163, Piscataway, NJ, USA, 2019. IEEE Press.

Rocha, R. C. O., Petoumenos, P., Wang, Z., Cole, M., and Leather, H. Effective function merging in the ssa form. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2020, pp. 854–868, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450376136.

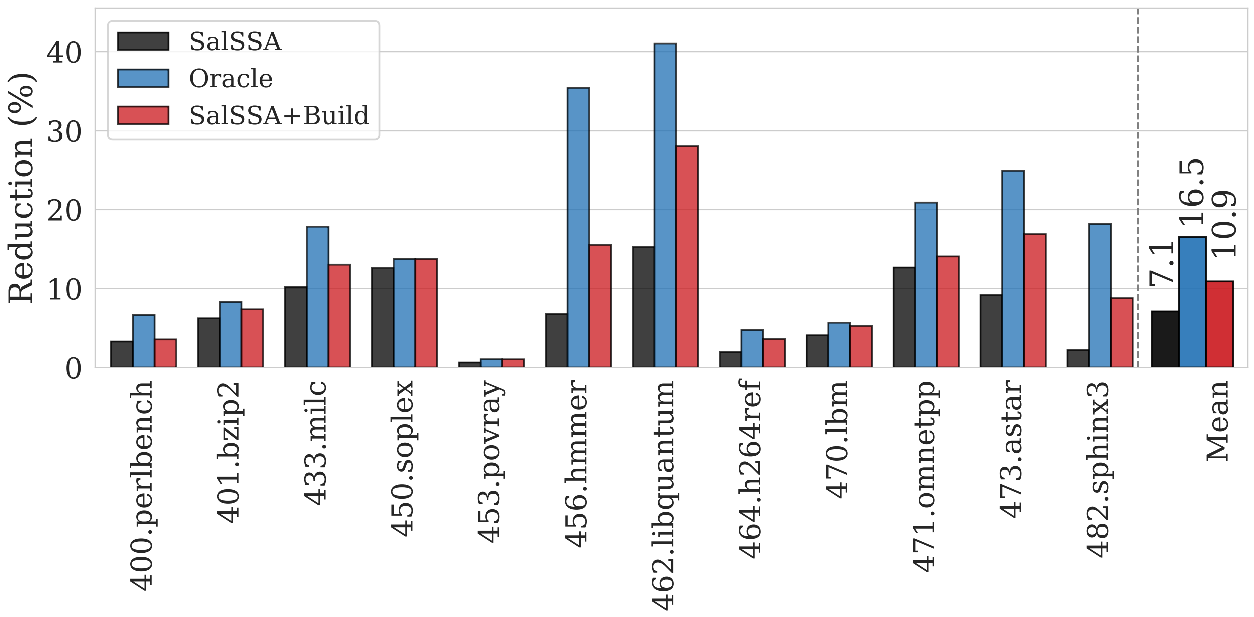

Figure 5. .

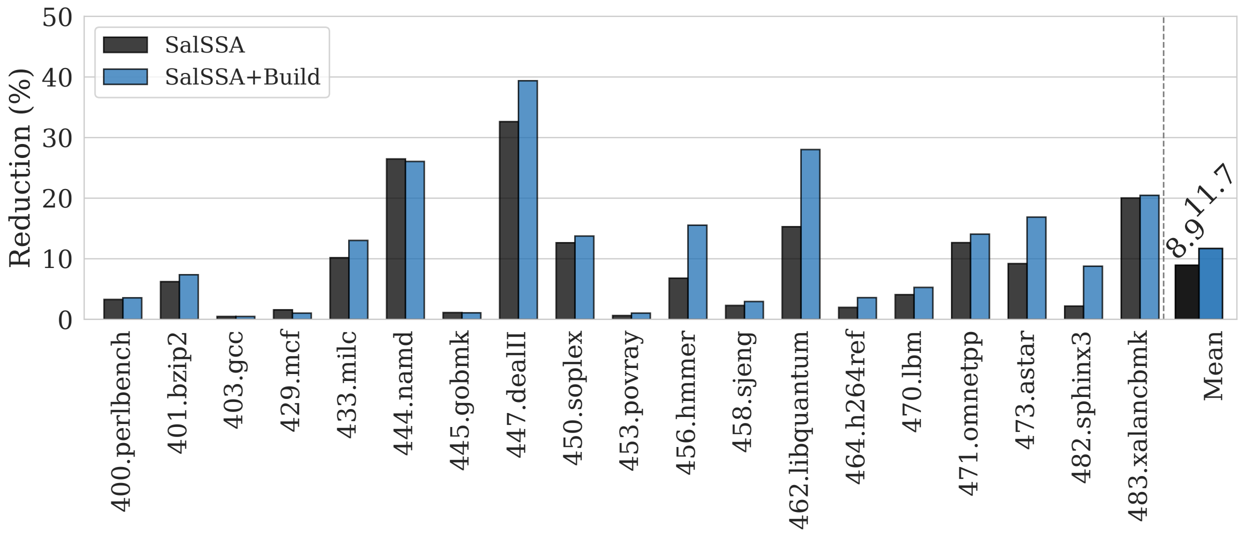

Figure 6. .