Function Merging by Sequence Alignment (FMSA)

Rocha, R. C., Petoumenos, P., Wang, Z., Cole, M., & Leather, H.
In 2019 IEEE/ACM International Symposium on Code Generation and Optimization (CGO '19)

Presenters: (Group 19) Jiaxing Yang, Yuzhou Mao

Motivation

- Reducing code size is important for resource constrained systems

https://developer.amazon.com/blogs/alexa/post/2a32d792-d471-4136-8262-79962a2b4d72/cpu-memory-and-storage-for-alexa-built-in-devices

Background

- Limitations of state-of-the-art (SOTA) [1] and LLVM's identical merging technique [2]

```
glist_t glist_add_float32(glist_t g, float32 val){
  gnode_t *gn;
  gn = (gnode_t *) mymalloc (sizeof(gnode_t));
  gn->data.float32 = val;
  gn->next = g;
  return ((glist_t) gn);
}

glist_t glist_add_float64(glist_t g, float64 val){
  gnode_t *gn;
  gn = (gnode_t *) mymalloc (sizeof(gnode_t));
  gn->data.float64 = val;
  gn->next = g;
  return ((glist_t) gn);
}
```

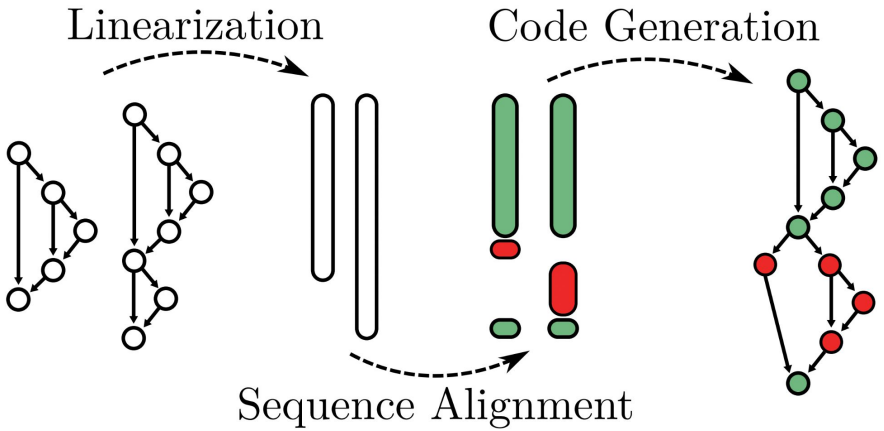
https://cmusphinx.github.io/doc/sphinxbase/glist_8c_source.html

[1] Tobias J.K. Edler von Koch, Björn Franke, Pranav Bhandarkar, and Anshuman Dasgupta. Exploiting Function Similarity for Code Size Reduction. (LCTES '14)

[2] https://llvm.org/docs/MergeFunctions.html

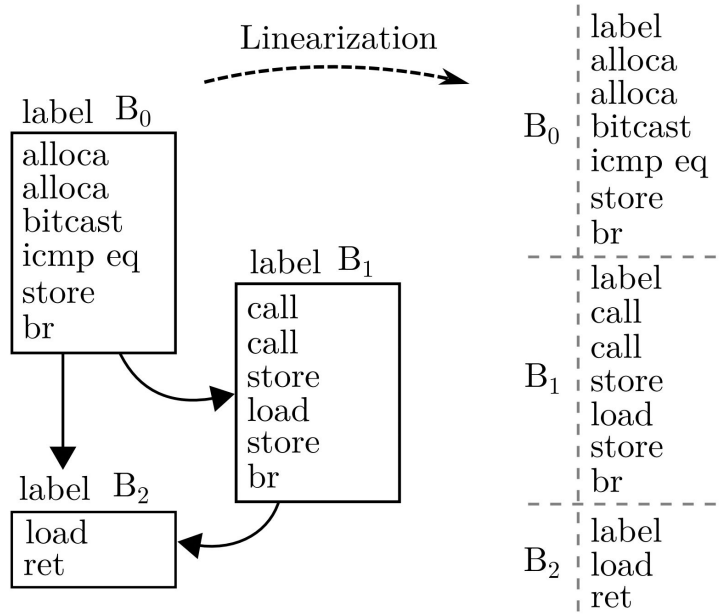
Approach - Overview

- Linearize each function
  - Represent CFG as a sequence of labels and instructions
- Apply a sequence alignment algorithm
  - From bioinformatics
  - To identify similar regions
- Code generation
  - To generate the merged function



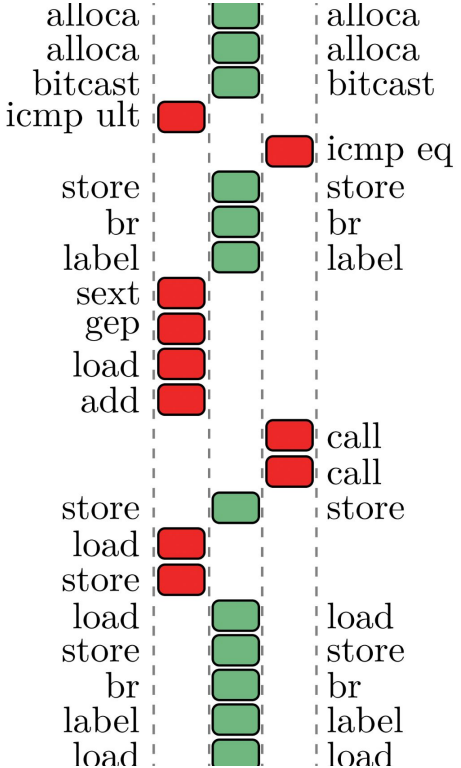
Approach - Linearization

- Traverse the CFG
  - For each block, output the label and instructions
- Maintain the original order of instructions inside a single block
- Edges are represented by branches + labels



Approach - Sequence Alignment

- SA is widely used in molecular biology
  - Identify similar DNA subsequences
- Needleman-Wunsch algorithm is used
  - Optimal with a given scoring system
  - Based on dynamical programming



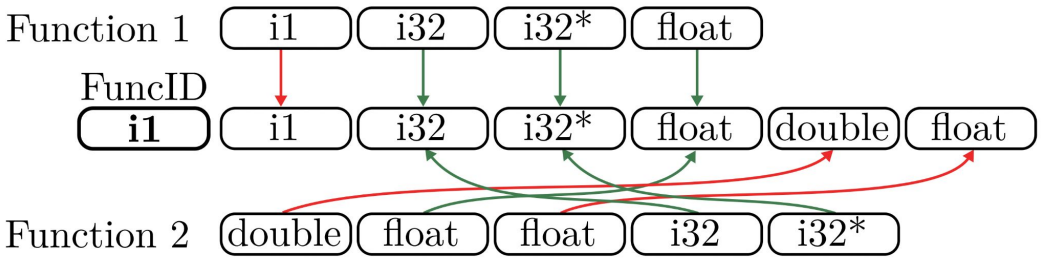
Approach - Equivalence Evaluation

- Equivalence between instructions
  - Semantically equivalent opcodes
  - Equivalent types
    - Can be losslessly bitcasted
- Equivalence between labels
  - Labels of normal basic blocks are ignored

Approach - Code Generation

To maintain the semantics of the original functions, we must be able to pass the parameters to the newly merged function

- Function identifier is needed to guard blocks from different functions
- The original parameters need to be merged
  - Order is unimportant
- Try to reuse parameters
  - Reduce overhead
  - Avoid *select* instructions



Approach - Code Generation

Return types also need to be merged.

- Two non-void return types
  - Use the larger one as base
  - Bitcast return values as the base
  - Reverse on the caller side
- One void return type
  - Directly return the non-void type
  - The return value will be discarded by the caller

Approach - Code Generation

Two passes to generate merged function

- First pass creates basic blocks and instructions
  - Keep a mapping: original instructions and labels -> corresponding merged values
  - Create extra basic blocks and branches to maintain the semantics
- Second pass assigns operands and connect blocks
  - Use the mapping from previous pass
  - Use *select* instruction to choose values from different functions

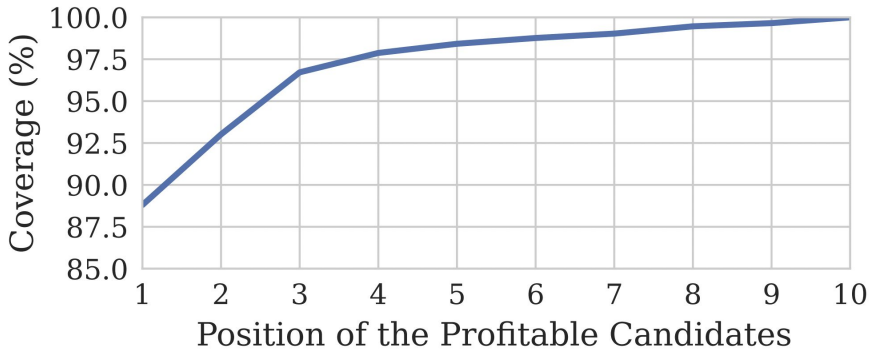
```
glist_t glist_add_float32(glist_t g, float32 val){
  gnode_t *gn;
  gn = (gnode_t *) mymalloc (sizeof(gnode_t));
  gn->data.float32 = val;
  gn->next = g;
  return ((glist_t) gn);
}

glist_t glist_add_float64(glist_t g, float64 val){
  gnode_t *gn;
  gn = (gnode_t *) mymalloc (sizeof(gnode_t));
  gn->data.float64 = val;
  gn->next = g;
  return ((glist_t) gn);
}
```

Merged Function

```
glist_t merged(bool func_id,
               glist_t g, float32 v32, float64 v64){
  gnode_t *gn;
  gn = (gnode_t *) mymalloc (sizeof(gnode_t));
  if (func_id)
    gn->data.float32 = v32;
  else
    gn->data.float64 = v64;
  gn->next = g;
  return ((glist_t) gn);
}
```

Profitable Functions

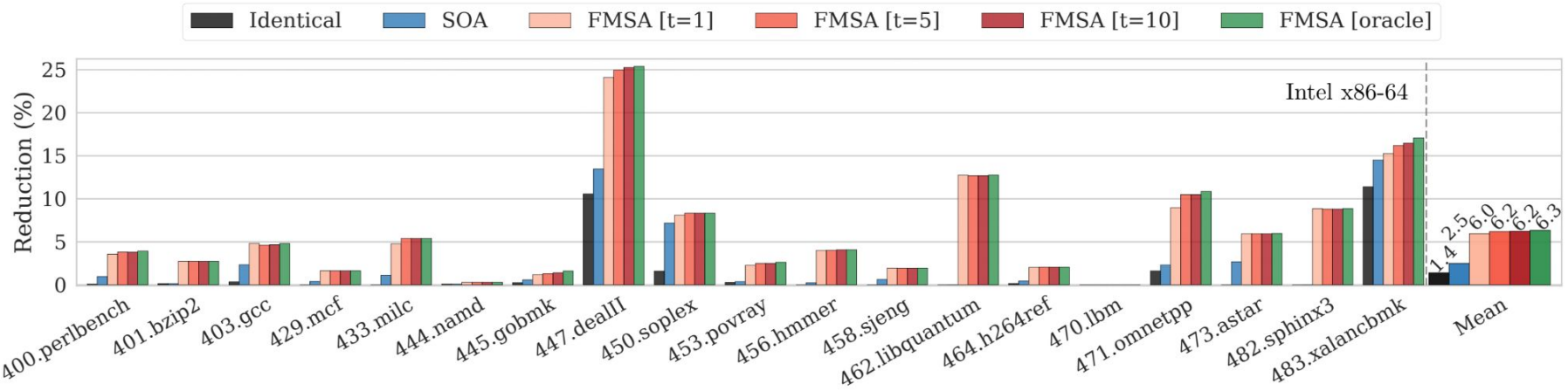
- Estimate similarity of two functions
  - Opcode frequencies
  - Type frequencies (return type / parameter type)



(Position in a priority queue ordered by similarity)

Experiments

- C/C++ SPEC CPU2006 (results below) and MiBench



Code size reduction for the linked object compared to baseline (no function merging)

Conclusion and Future Work

- Strengths
  - Addresses limitations of SOTA and outperforms by 2.4x on average
- Weaknesses
  - Larger compilation overhead
  - No improvement on some benchmarks
- Future directions
  - reduce compilation overhead
  - further code size reduction on some benchmarks

Questions?

Thanks!