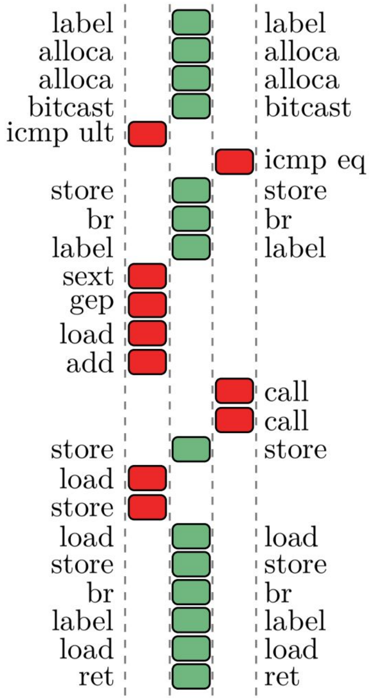
Speeding up FMSA

Liu Jiang, Leiqi Ye, Ying Yuan, Franklin Kong

Recap of FMSA

- Function merging by sequence alignment
- Borrowed the idea from biology
- Is able to merge two arbitrary functions
- Filters functions by fingerprint (basically opcode counts and types)



Problem with FMSA

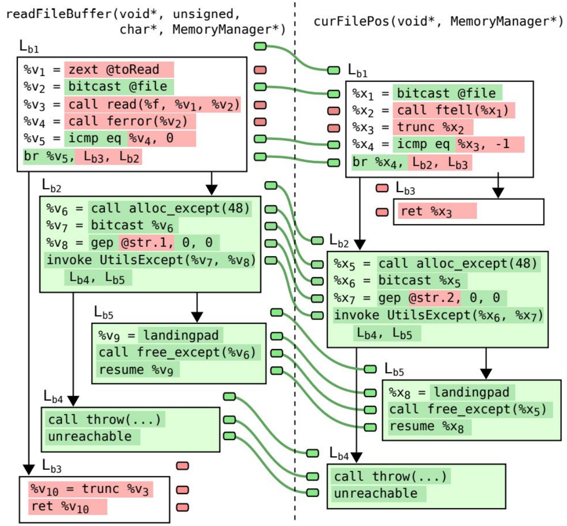
- Pairwise comparison for all available functions, which is O(n^2)
- Unavoidable, Manhattan distance is between two vectors
- From HyFM, only similar functions worth merging
- A lot of useless comparisons

```
mergeFunctions(Funcs):
    Worklist = Funcs
    AvailableFuncs = Funcs

    while worklist is not empty:
        F1 = Worklist.pop()
        sort AvailableFuncs by Manhattan distance
        for i in 1...MaxExploration:
            F1 = AvailableFuncs[i]
            FMerged = mergeBySequenceAligment(
            if FMerged.size < F1.size + F2.size:
                update call graph
                delete F2 from AvailableFuncs
                add FMerged to Worklist
```

Key Observation in FMSA

The most of the benefit of function merging often comes from merging highly similar function.**



Keyword:

- Tradeoff between code size and speed
- Linear-time comparison
- Cosine similarity | Manhattan distance
- Skip small functions

Solution: reference vector

- Only similar functions are worth merging, and they should have similar opcode counts
- Step:
  - Find a reference to compute Manhattan Distance
    - Reference vector: takes mean on each feature
  - Every function compute a value relative to the reference vector
  - Group similar functions together and merge within the group

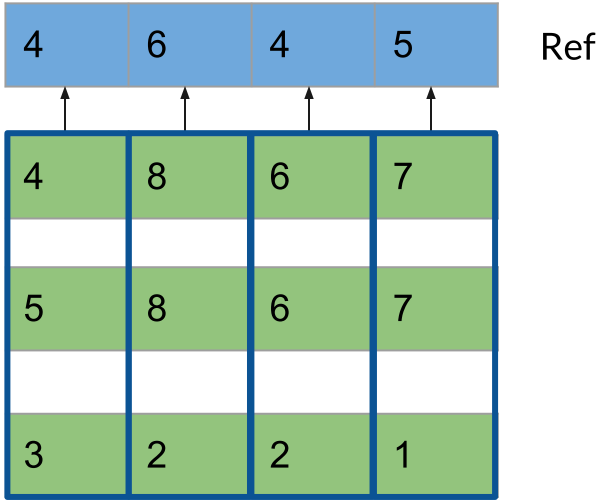
4	8	6	7

5	8	6	7

3	2	2	1

Solution: reference vector

- Only similar functions are worth merging, and they should have similar opcode counts
- Step:
    - Find a reference to compute Manhattan Distance
        - Reference vector: takes mean on each feature
    - Every function compute a value relative to the reference vector
    - Group similar functions together and merge within the group



Solution: reference vector

- Only similar functions are worth merging, and they should have similar opcode counts
- Step:
  - Find a reference to compute Manhattan Distance
    - Reference vector: takes mean on each feature
  - Every function compute a value relative to the reference vector
  - Group similar functions together and merge within the group

4	6	4	5	Ref
4	8	6	7	6
5	8	6	7	7
3	2	2	1	11

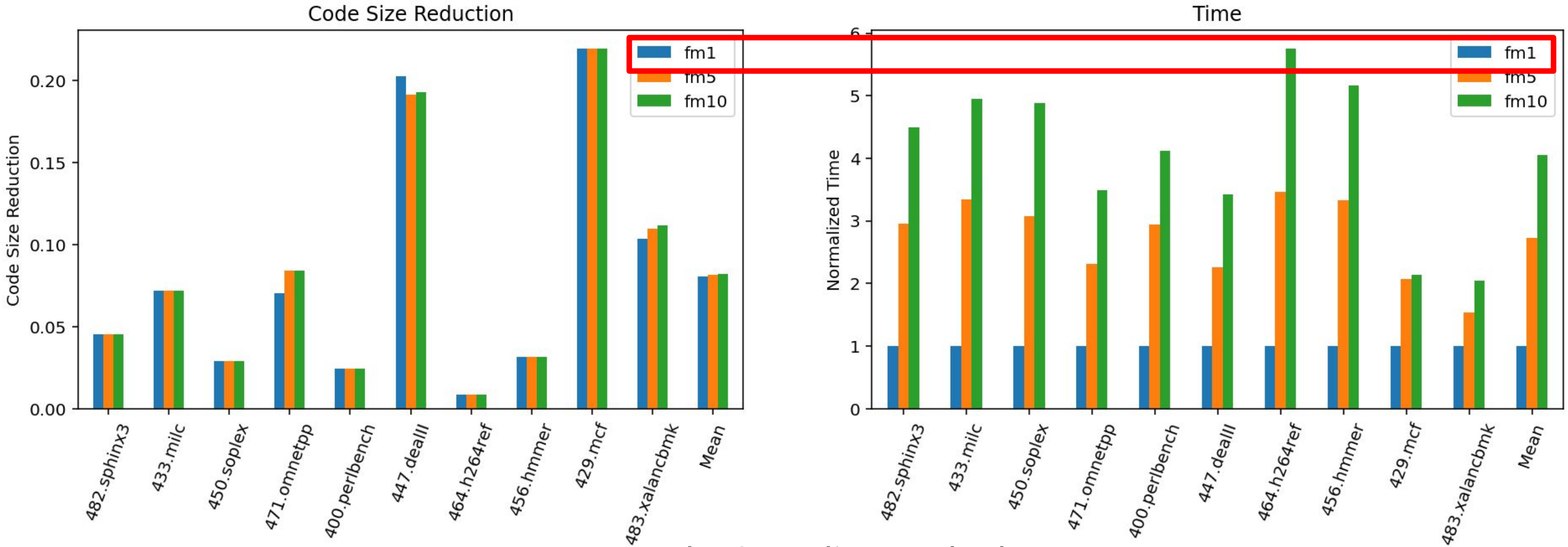
Solution: reference vector

- Only similar functions are worth merging, and they should have similar opcode counts
- Step:
  - Find a reference to compute Manhattan Distance
    - Reference vector: takes mean on each feature
  - Every function compute a value relative to the reference vector
  - Group similar functions together and merge within the group

4	6	4	5	Ref
4	8	6	7	6
5	8	6	7	7
3	2	2	1	11

[6, 7.5), [11, 12.5), ...

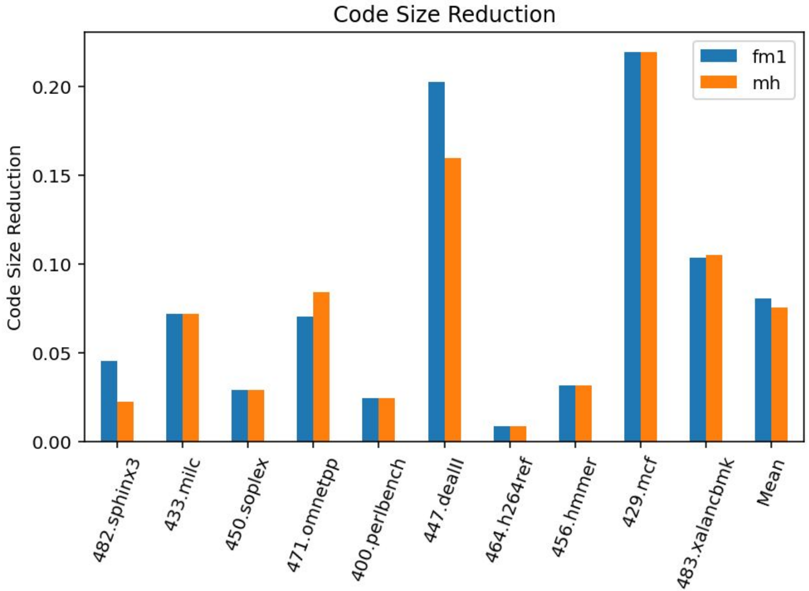
Experiment Setup

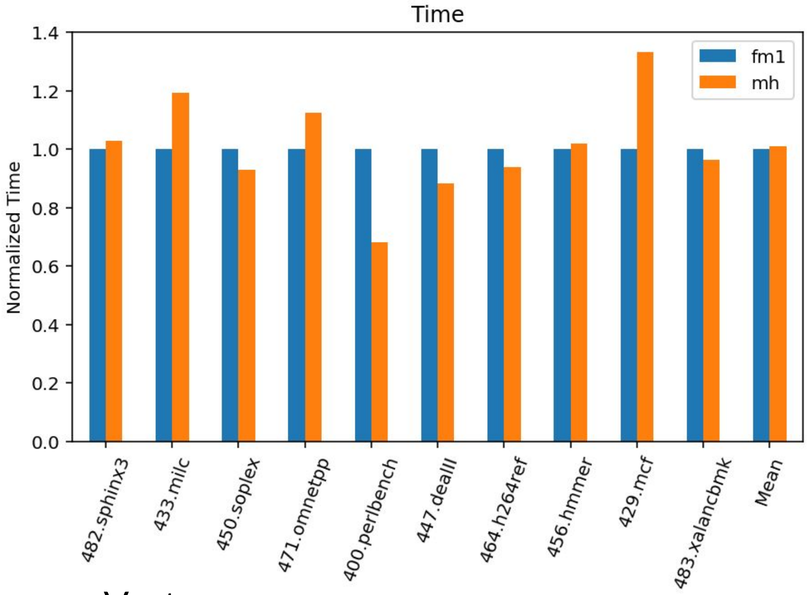


Result of Baseline Method

Result

Mean:
↑ 1% time
↓ 8% code size reduction





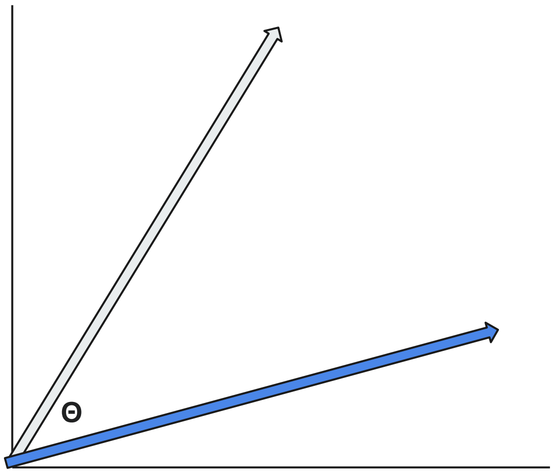
Result of Reference Vector

Potential problem with Manhattan Distance

4	4	4	4	Ref
5	7	8	10	14
5	6	8	10	13
1	1	1	1	12

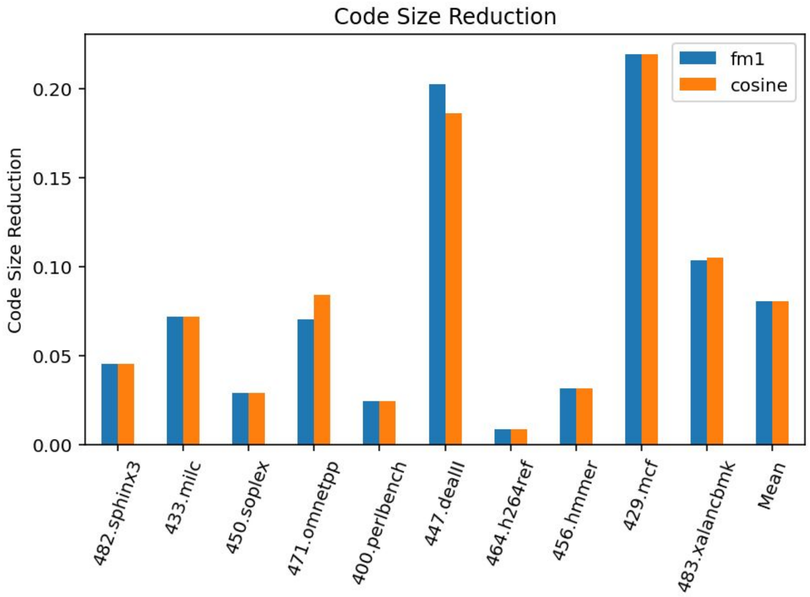
Cosine Similarity

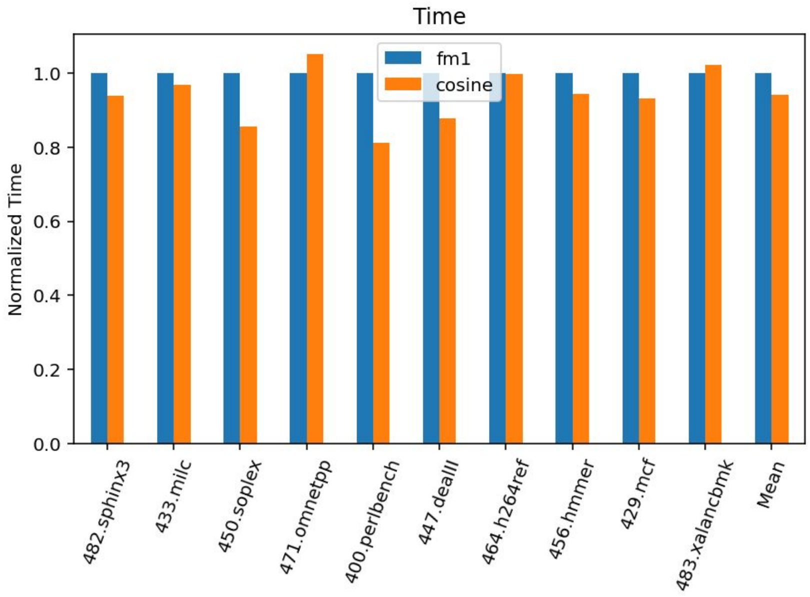
- Borrowed idea from Google Search
- Word counts of documents - Opcode counts of functions
- Cosine similarity
- Measures the angle, not the magnitude
- Smaller the angle, higher the similarity



Result

Mean:
↓ 6% time
- same code size reduction





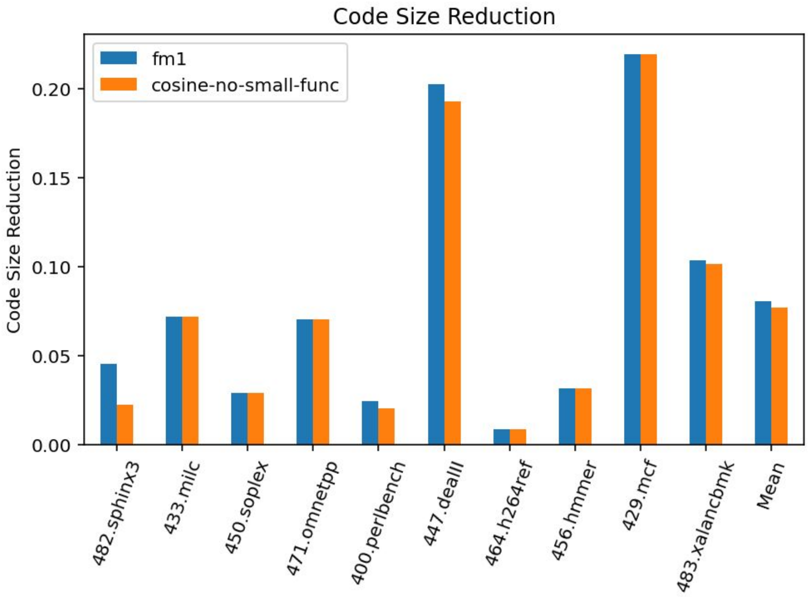
Skip Small Functions

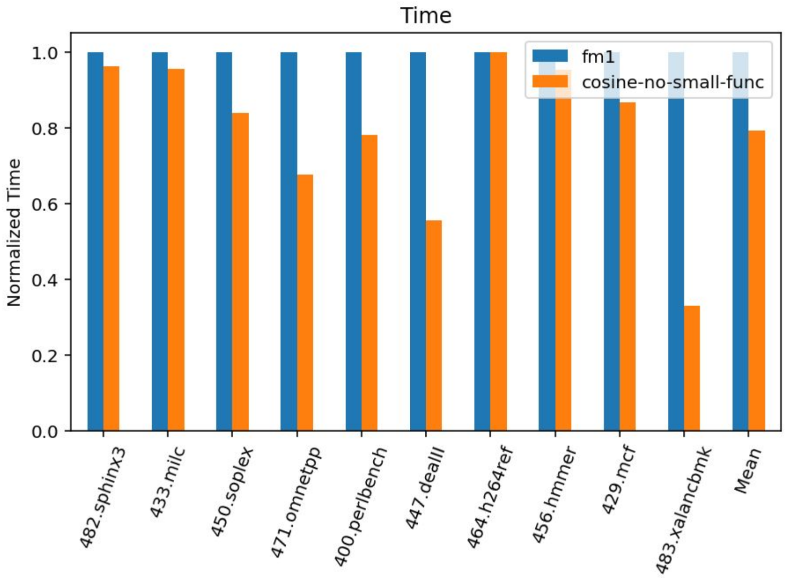
Sizes: 6 + 6<= 13?
Sizes: 4 + 2 <= 9?
⋮

- A lot of attempts on small sized functions
- IR cost model is not perfect
  - Function calls have costs
  - Though profitable on IR level, doesn't help in final code size
- Let's just skip small functions

Result

Mean:
↓ 21% time
↓ 5% code size reduction





Summary

- Speed up FMSA at the cost of less code size reduction
- Avoid pointless comparison
- Cosine similarity
- Skip small functions