Object Inlining

Ondřej Lhoták
November 12, 2001
308-762

Reference

- Julian Dolby, Andrew A. Chien. *An Automatic Object Inlining Optimization and its Evaluation*. PLDI 2000

Outline

- Introduction
- A bit about the analysis
  - Greek letters omitted
- Experimental evaluation
- Conclusion

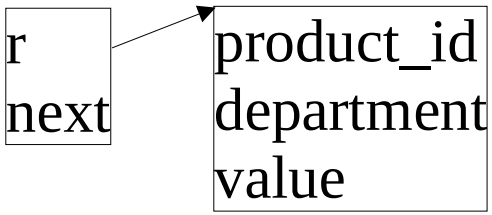
Problem

```
class List {
    Record      r;
    List        next;
}
```

```
class Record {
    int         product_id;
    int         department;
    float       value;
}
```

C++

Java



ICC++ and Concert Compiler

- ICC++: "a language with C++-like syntax and Java-like semantics"
  - Compiles C++ programs
  - Turns all embedded objects into references, like Java
- Concert compiler: "iterative adaptive analysis"

One-to-One Fields

- Definition: f is a *dynamic one-to-one field* if for a given execution, every parent object corresponds to exactly one child object through f.

Dynamic Field Inlining

- Definition: A *dynamic field inlining* of a one-to-one field is a substitution of parent objects for child objects wherever they appear in the execution.

Proof

- The paper gives a 1-page proof that:
  - Program is equivalent after object inlining has been applied to a one-to-one field
  - After inlining, the inlined field is no longer needed

Finding one-to-one fields

- Verifying common creation
  - parent and child must be created in same *inter-procedural control dependence region*
- Verifying data flow
  - check that the child creation and child assignment sites *must be aliased*

Finding field uses

- Must find *all* uses of the field
  - uses before it was assigned to parent
  - uses after it was assigned to parent

Transformation

- Build fused classes based on inlinable fields
- Replace object creations with fused object creations
- Replace child state accesses with fused object accesses

Building fused classes

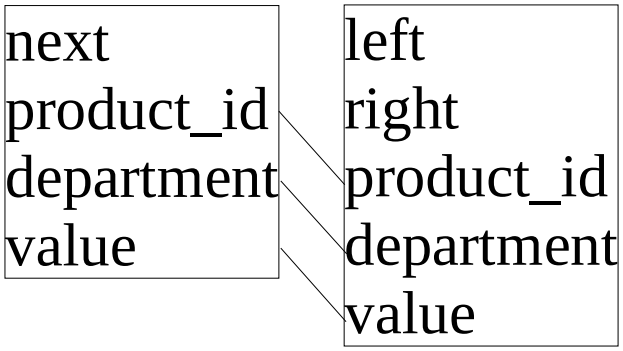
- Fused object classes combine not only data, but also methods
- If methods from child class are to be shared, data layout must be the same in fused classes containing the child class

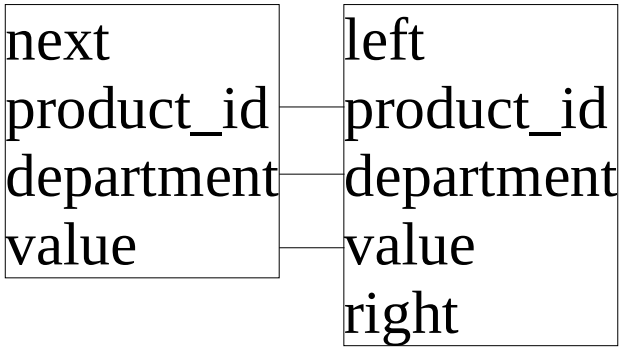
Example

```
class List {
    List next;
    Record r;
}
```

```
class Record {
    int   product_id;
    int   department;
    float value;
}
```

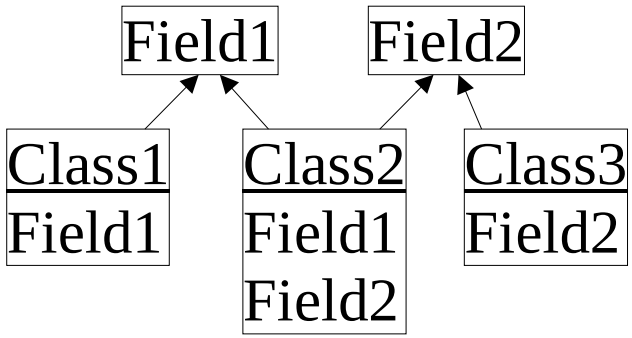
```
class Tree {
    Tree left, right;
    Record r;
}
```





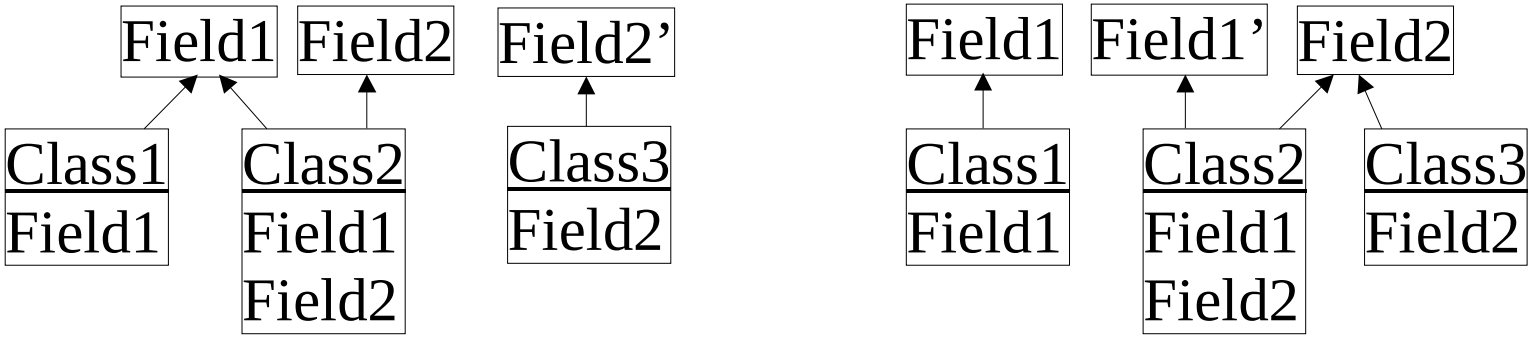
Approach

- Arrange classes into trees (like single-inheritance hierarchy)
- Conflicts can arise only if two fields appear in overlapping sets of classes, neither a subset of the other.



Solution

- Try all forests



- Take the one that minimizes code growth

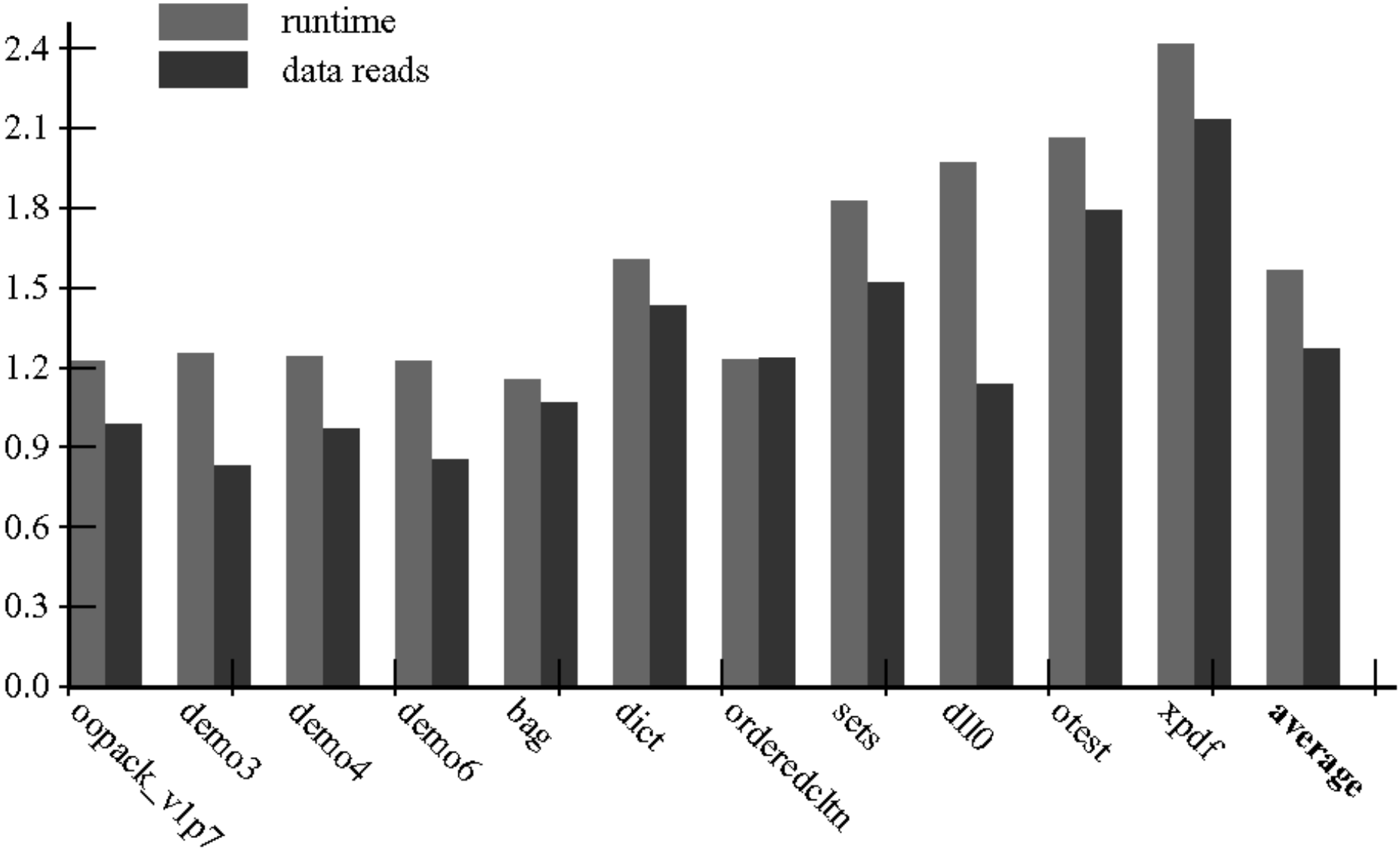
Evaluation

benchmark	lines	inlinable data structures
oopack	760	complex numbers
AI library (3000 lines) programs		
demo3	300	lists, search nodes, arrays
demo4	300	lists, search nodes, arrays
demo6	200	lists, arrays
NIHCL library (20000 lines) programs		
bag	100	bag, set, array, iterator
dict	100	assoc, bag, set, array, iterator
orderedcltn	100	collections, array, iterator
sets	100	set, array, iterator
OATH library (18000 lines) programs		
dll0	150	arrays, smart pointers
large programs		
otest	30000	lists, arrays, wrappers, parser
xpdf	25000	streams, arrays, child

Evaluation

	Fields				
		Inlinable			
Benchmark	*All*	*Hand*	*Precise*	*1-CFA*	*%*
oopack_v1p7	4	0	0.5	0	12
demo3	13	3	5	0	38
demo4	14	3	6	0	42
demo6	14	5	6	0	42
dll0	17	2	2.74667	1	16
bag	30	6	11.2022	8.5	37
dict	31	6	11.4189	6.5	36
orderedcltn	27	4	10.4732	7.5	38
sets	27	4	10.381	7.5	38
otcst	38	5	15.7167	10.5	41
xpdf	133	54	45.25	23.4348	34

Performance vs. g++



Performance

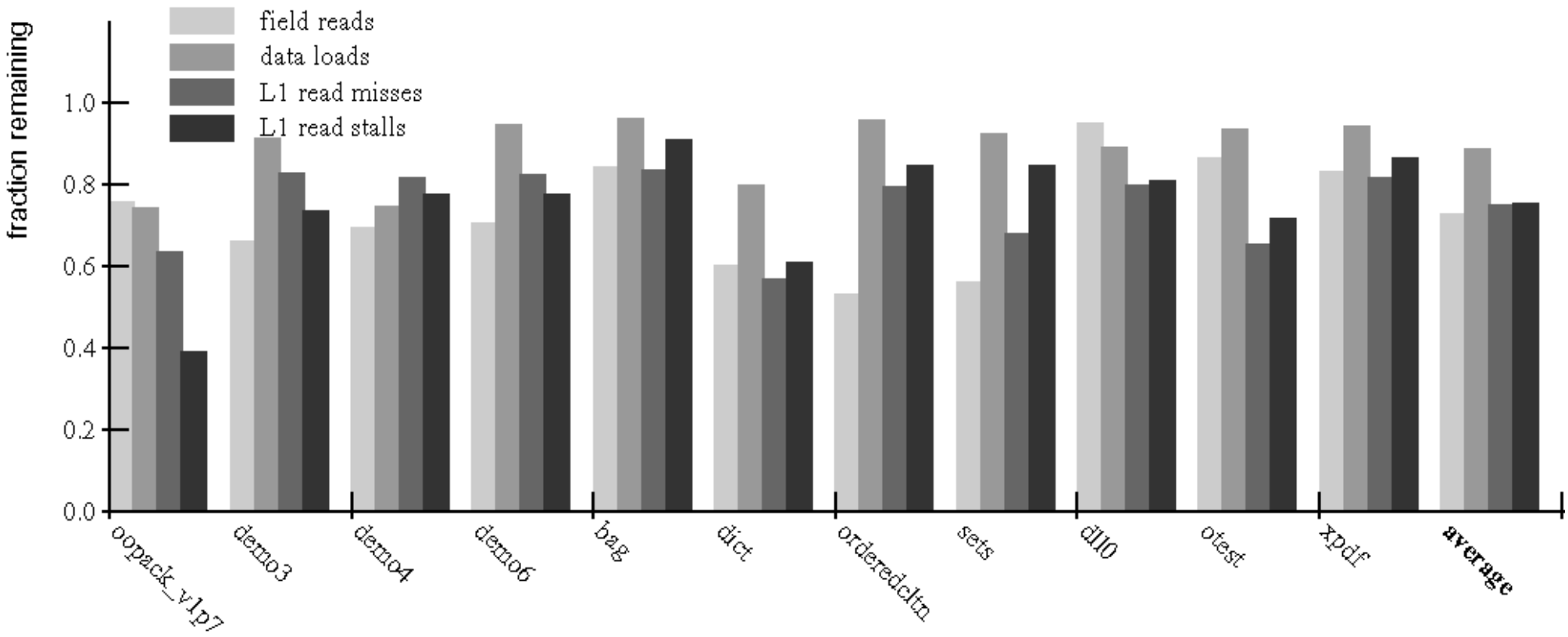
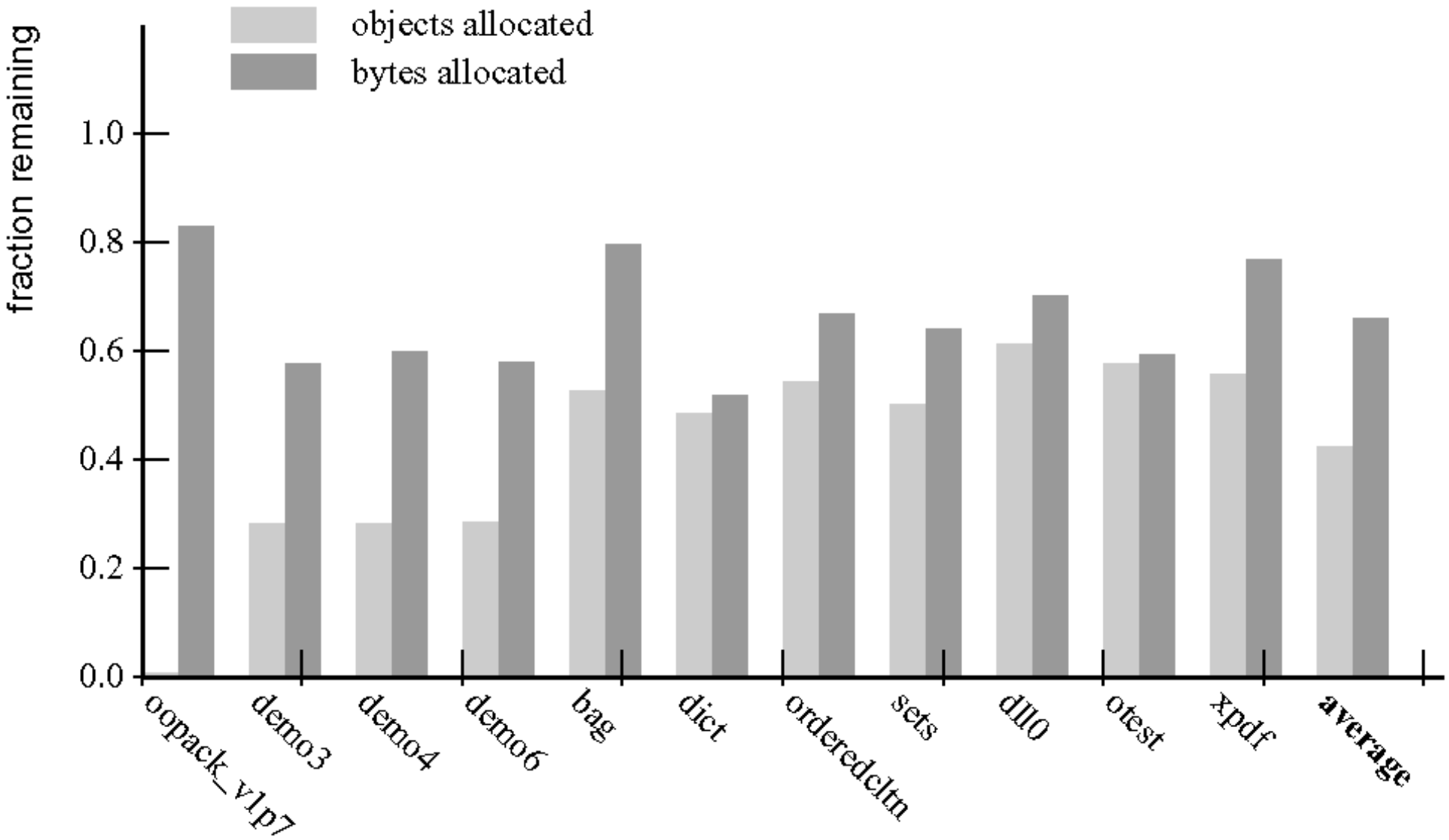
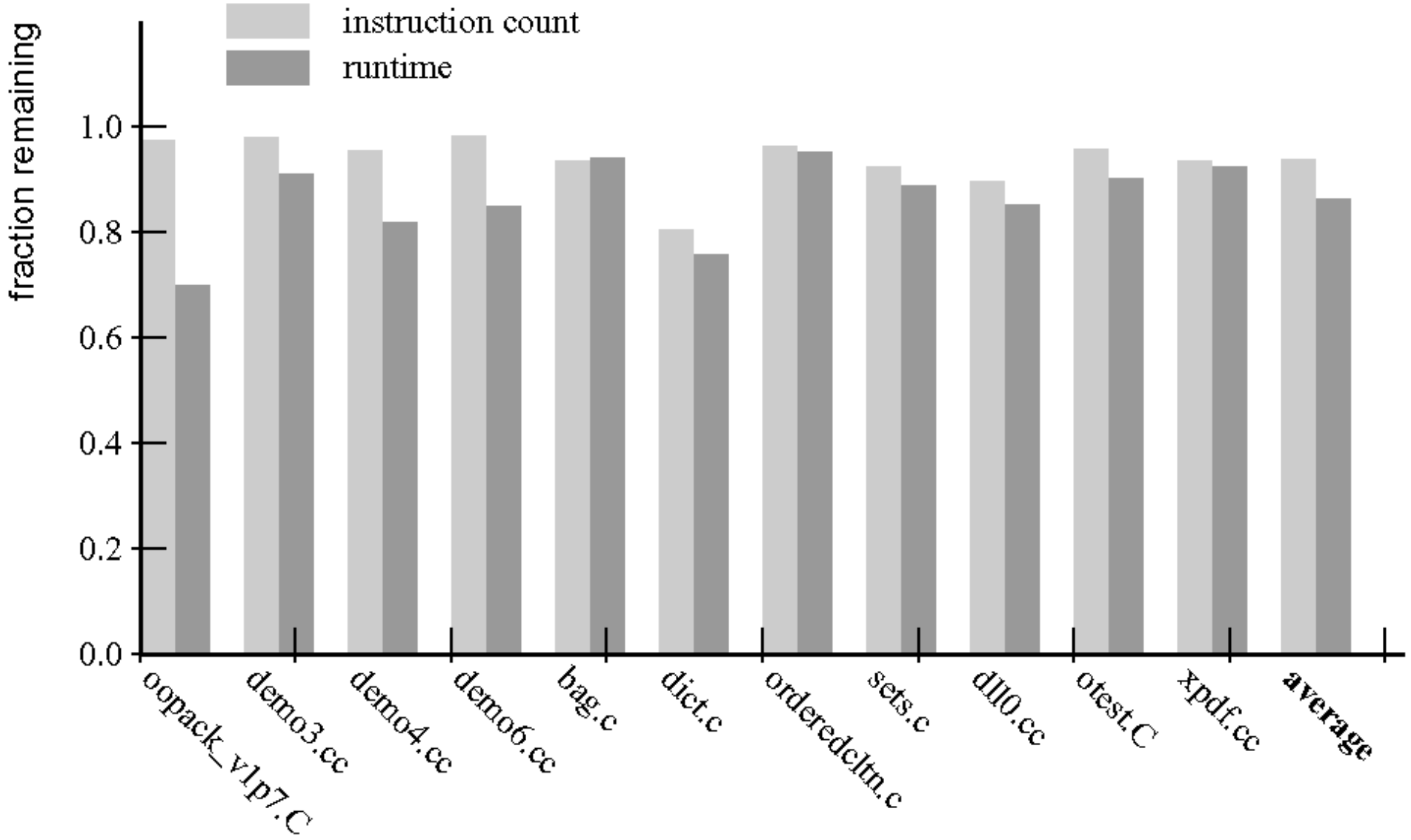


Figure 16: Field Read Counts and L1 cache misses with and without Object Inlining

Performance



Performance



Conclusions

- Significant performance improvement
- Analysis requires "incremental adaptive" implementation to be fast enough
- How would it do on Java?
  - Probably more object inlining opportunities
  - Probably harder to find them