A Comprehensive Approach to Array Bounds Check Elimination for Java

Feng Qian, Laurie Hendren, Clark Verbrugge

Sable McGill

Sable Research Group
School of Computer Science, McGill University
Montreal, CANADA

http://www.sable.mcgill.ca

Introduction

- Array bounds checks are required by the semantics of the Java programming language.
- Runtime checks can slow down the program execution significantly.
- This paper presents an algorithm for proving which array references are safe (do not need bounds checks).

The Problem

- *ArrayIndexOutOfBoundsException*.
- Precise exception requirements limit code motion and loop transformations.
- A multi-dimensional array is an array of arrays (can be ragged).
- The lower bound check comes free with the upper bound check on most modern architectures.

Overview

- Intra-procedural, flow-sensitive analysis: *Variable Constraint Analysis* (VCA).
- Two extensions:
  1. class level analysis: *array field analysis*.
  2. inter-procedural analysis: *rectangular array analysis*.

Difference Constraints

- Bounds checks for $a[i]$: $0 \leq i \leq a.length - 1$?
- From a program text, a static analysis is able to collect a set of inequalities.
- The system of difference constraints is represented as a constraint graph.
- The shortest path of the graph answers above query.

Variable Constraint Graphs

- Nodes: *int* type locals, *array* type locals, the constant 0 node.

- Edges: $e(u, v) = w$ represents the difference constraint $v - u \leq w$.

- Ordering of edge weights: $\perp \sqsubset minint \sqsubset \ldots \sqsubset c \sqsubset c + 1 \sqsubset c + 2 \sqsubset \ldots \sqsubset maxint \sqsubset \top$

An Example of VCG

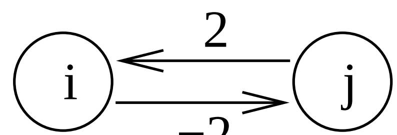
$s_0 : i = j + 2;$

$s_1 : a[i] = \cdots;$

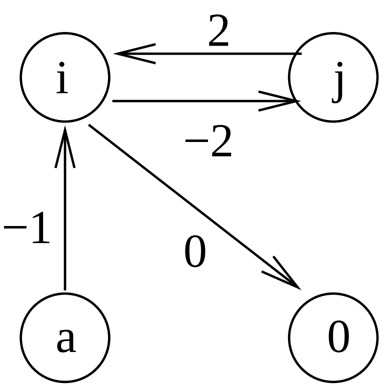
$s_2 : i = foo(\cdots);$

$s_3 : a[j] = \cdots;$

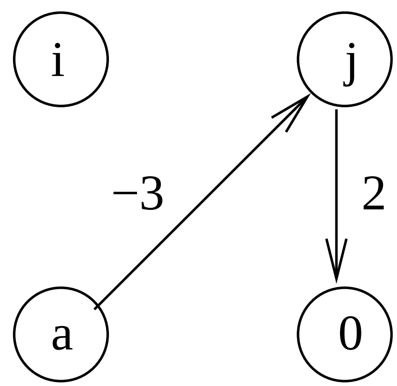
(a) a basic block



(b) vcg after s_0



(c) vcg after s_1



(d) vcg after s_2

VCG Properties

- Weighted directed edges.
- Inequalities are transitive.
- The shortest path gives the tightest constraint.
- But, it cannot represent multiplication and division.

Array-Related Liveness Analysis

- Specialized live variable analysis focusing on only variables related to array references.
- Computes accurate node sets for VCGs at interesting program points.

Expr	Cond	Gen	Kill
a [i]		a, i	
a = new T[i]	a	i	a
i = j + c	i	j	i
. . .			

Variable Constraint Analysis (1)

- Forward, flow-sensitive, optimistic analysis.
- Break basic blocks at statements with array references.
- The analysis approximates VCG edge weights.
- At a join point, merging two graphs by choosing the MAX weight for each edge, then widening the edge weights, if necessary.

Variable Constraint Analysis (2)

- The flow function for a basic block:

  INPUT : the VCG at the entry of block.

  OUTPUT : the VCG after taking the effect of statements of the block.

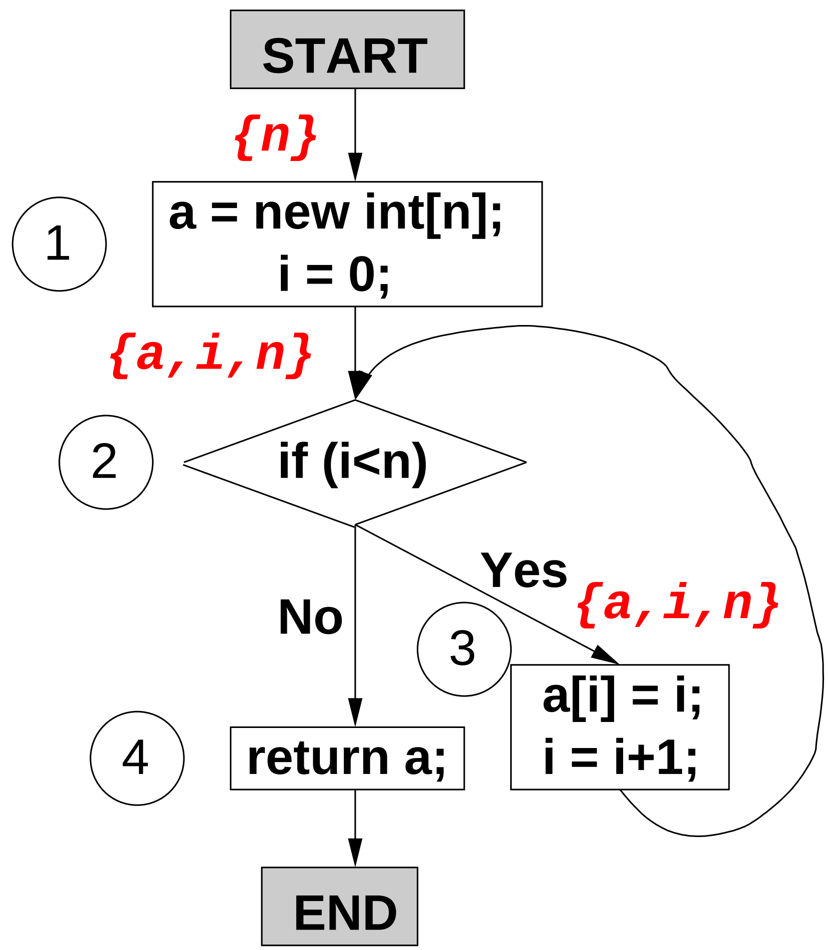
- Edge weights at the entry point are initialized to $\top$, others are set to $\perp$.

- Take conditions of branches into account (for *if (i<n)*: $i - n \leq -1$ on `true` path, $n - i \leq 0$ on `false` path.

Tune The Analysis

- Limit the size of constraint graphs by pre-computing the set of array-related live variables.
- Widen edge weights at the confluence points.
- Enforce a good ordering in the work list:
  - the loop body is visited before the loop exit;
  - inner loops reach a fixed point before outer loops.

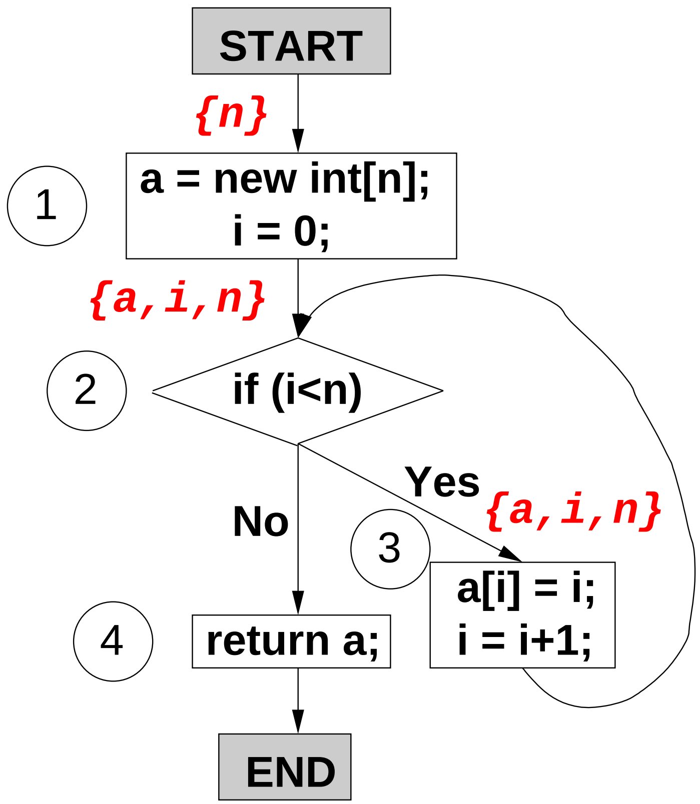
```
int[] init_array(int n)
{
    int[] a = new int[n];
    for (int i=0; i<n; i++) {
        a[i] = i;
    }
    return a;
}
```

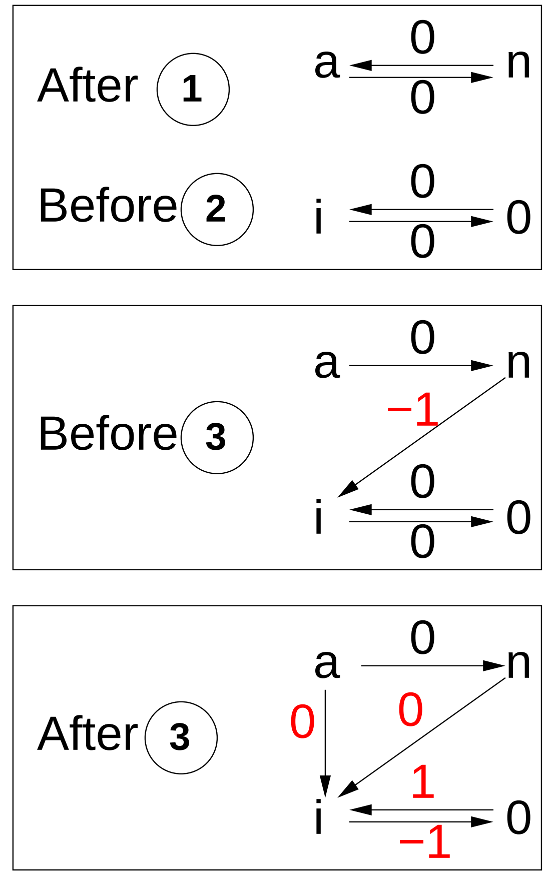


Step 1: order CFG nodes

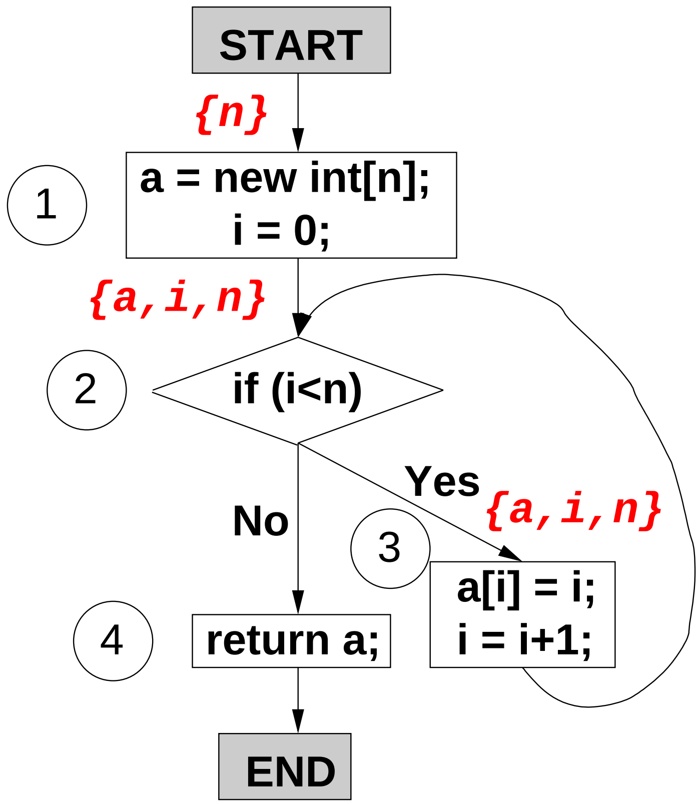
Step 2: array−related live variable analysis

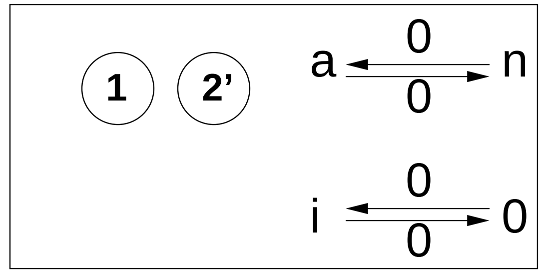
Step 3: Variable Constraint Analysis

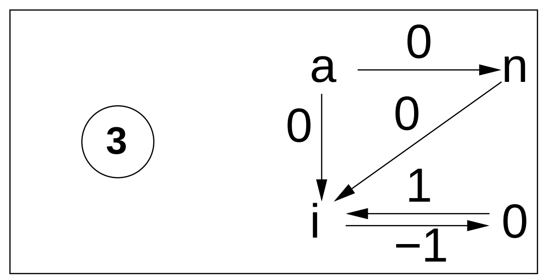


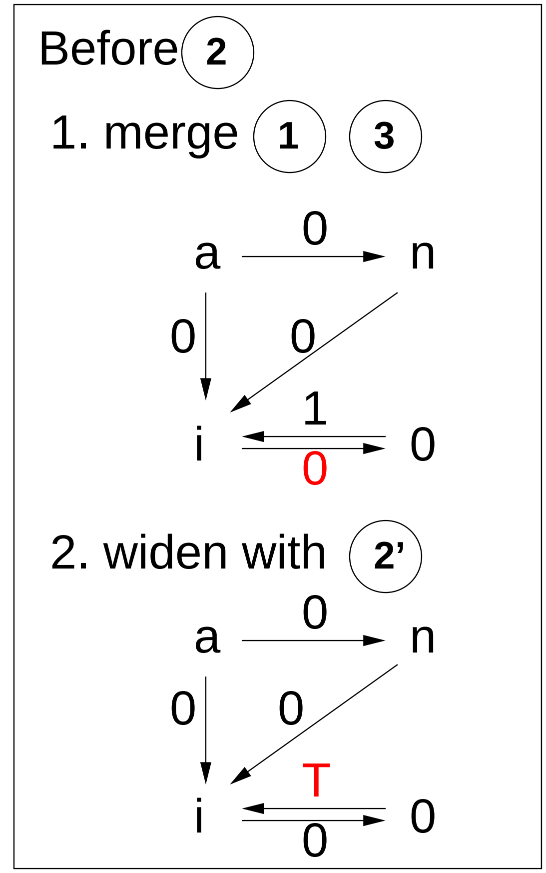


Step 3: Variable Constraint Analysis

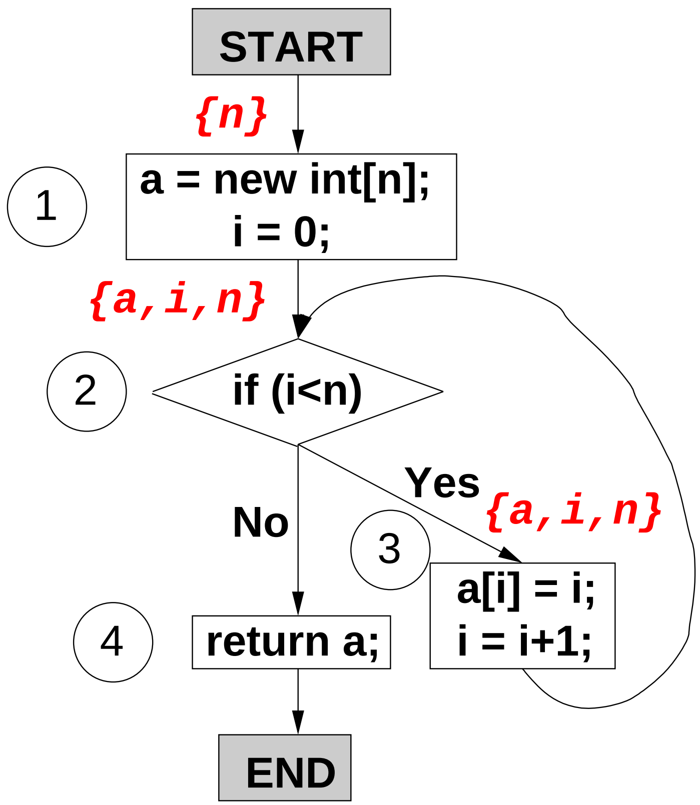


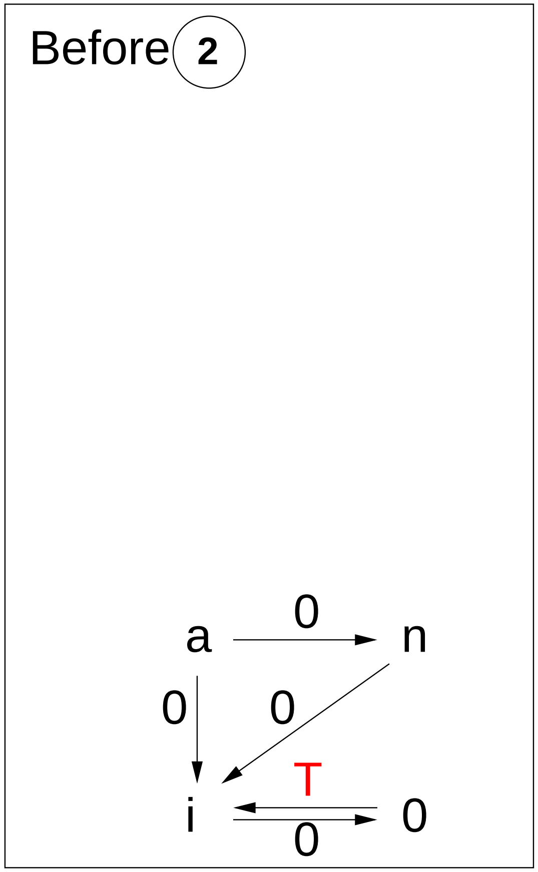




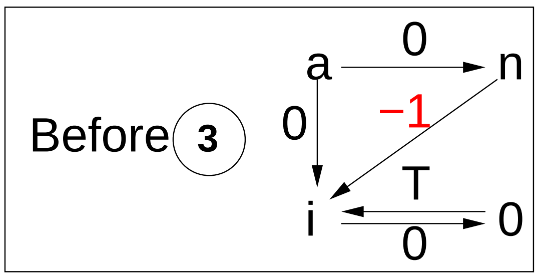


Step 3: Variable Constraint Analysis






```
i - a <= -1
0 - i <= 0
```



Extension 1: Array Field Analysis

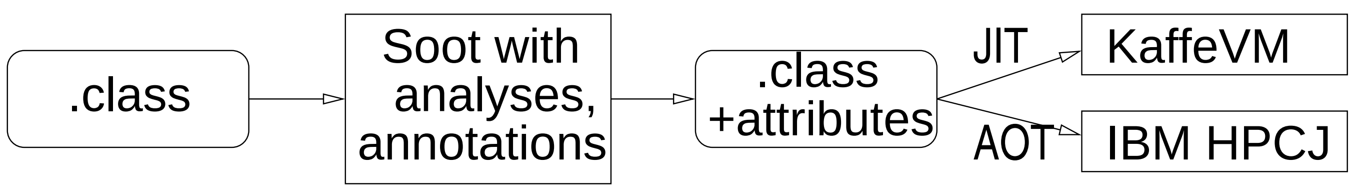
- Example: *final int[] m={2,4,7,9...};*
- Observation : a *final* or *private* field can only be assigned a value in the declaring class.
- Find all possible array lengths using DU/UD chains.
- Identify those array fields with constant lengths.
- The constant array length is used by the intra-procedural analysis.

Extension 2: Rectangular Array Analysis

- Array Type Graph:
  - a = multianewarray T[i][j] : $TRUE \longrightarrow a$;
  - a[i] = c : $FALSE \longrightarrow a$;
  - a = b : $a \longleftrightarrow b$.
- A variable 'a' has a rectangular shape if $TRUE \leadsto a$ and $FALSE \not\leadsto a$.
- A special node in VCG for the length of subarrys: $a[$.

Experimental Approach

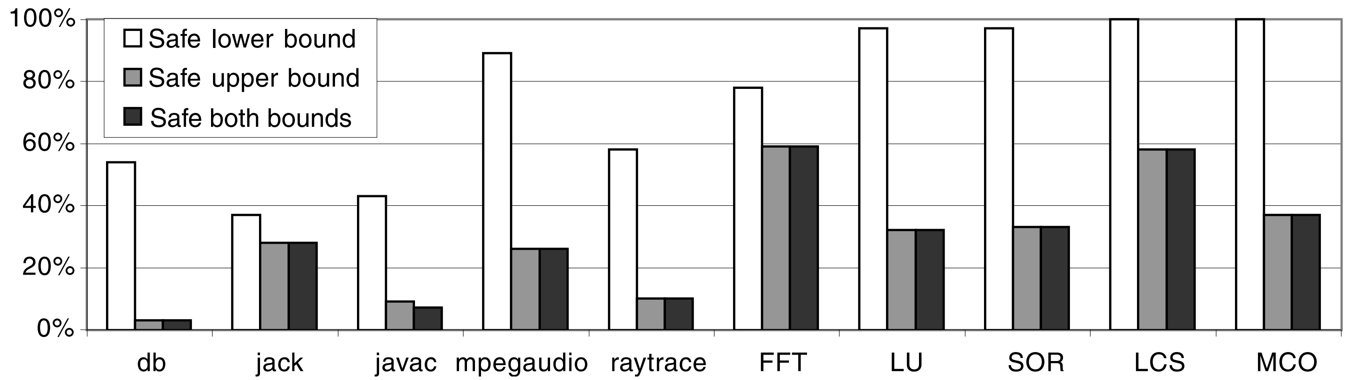
- Used as ahead-of-time analyses in the Soot framework.
- Analysis results encoded in class files as attributes.
- Modified Kaffe JIT and IBM HPCJ take advantage of such attributes.



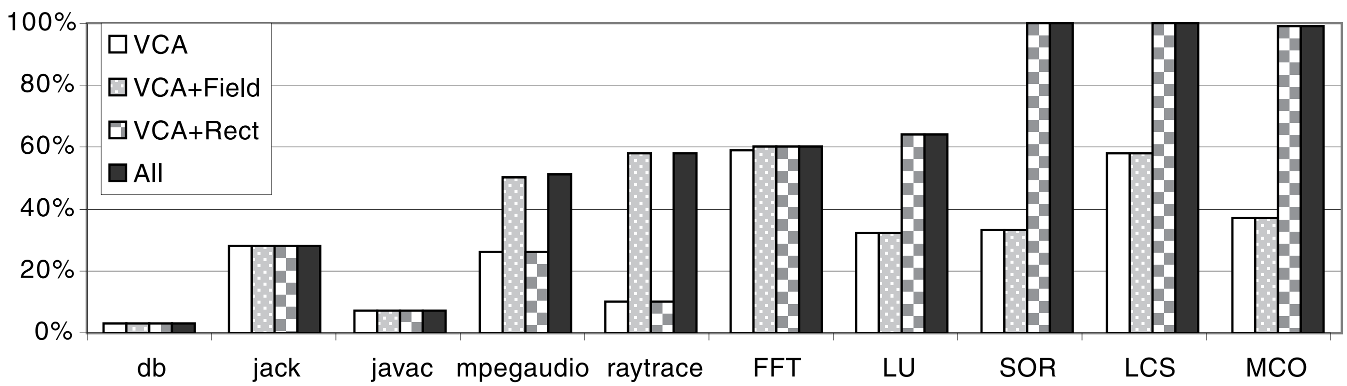
Characteristics of The Algorithm

	Graph size (avg)	Graph size (max)	Blocks	NonZero Blocks	Iter (avg)
db	3.17	6	280	89	1.28
jack	2.5	6	2076	1892	1.04
javac	2.45	6	3347	1631	1.27
mpegaudio	3.42	10	6987	6670	1.10
raytrace	2.56	6	626	476	1.31
scimark2	5.8	12	388	301	1.79
LCS	9	13	59	55	2.8
MCO	4.6	11	98	95	2.0

Analysis Results

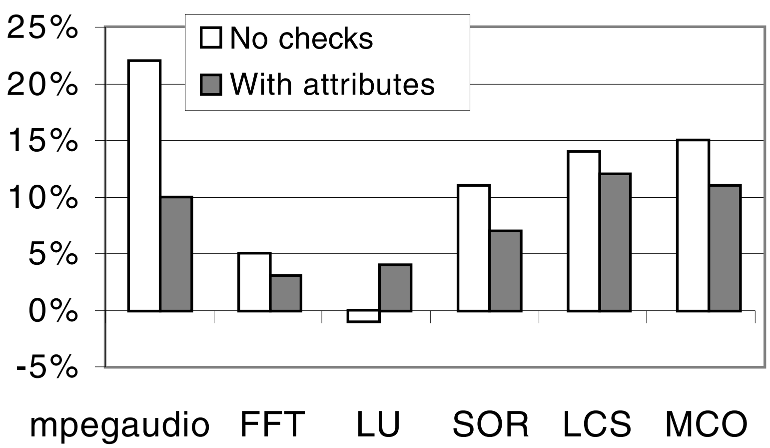


(a) Results of the intra-procedural analysis

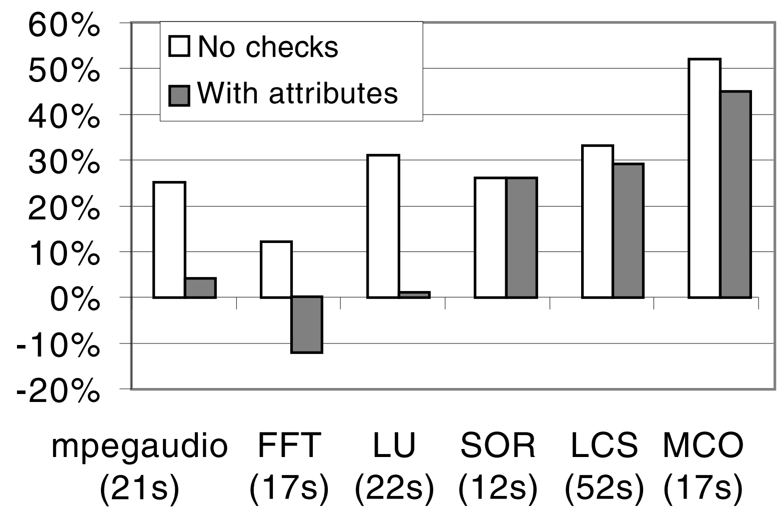


(b) Improvements due to field and shape analysis

Speedups in KaffeVM and HPCJ



(a) Kaffe VM



(b) HPCJ (other optimizations on)

Related Work (1): ABCD (PLDI'00)

- ABCD (R. Bodik et. al.) is based on an extended SSA form, and uses one constraint graph for a method; VCA is based on CFG, and computes small program-point specific constraint graphs.
- ABCD proves safe bounds on demand; VCA analyzes all references at once.
- ABCD is capable of catching partial redundant bounds checks; VCA is not.
- VCA analyzes lower and upper bounds at the same time, and the results can be improved by two extended analyses.

Related Work (2): Detect Array Memory Leaks (CC'00)

- R. Shaham et. al. used very similar representations as VCG to detect live ranges of array reference.
- The algorithm works on supergraphs of particular library classes (Vector).
- VCA focuses on intra-procedural analysis and uses various techniques to reduce the cost of data-flow analysis.

Conclusions

- We presented a collection of analyses for eliminating array bounds checks in Java.
  - Variable Constraint Analysis
  - Array Field Analysis
  - Rectangular Array Analysis
- Techniques reduce the cost of the data-flow analysis.
- The algorithm is effective for proving safe array references.