

Adaptive Online Context-Sensitive Inlining

by Kim Hazelwood and David Grove – CGO03

Anh Vo

avo2@gmu.edu

Acknowledgement

- Part of these slides are taken (with permission) from Dr Hazelwood's presentation at CGO03
<http://www.cs.virginia.edu/kim>

Outline

- Background
- Problem/Motivation
- Algorithm/System overview
 - Implementation
 - Adaptive Policies
- Evaluation
- Related Work
- Critical Evaluation

Background – Why, Why Not, When

- **Why Inlining?**
 - Method call is expensive (saving/restoring active registers, updating return address, passing parameters...)
 - Missed optimization opportunities when code is broken up
 - Reduce code size

```
int foo (int x, int y) {  
    return foo2(x, y);  
}  
int foo2 (int x, int y) {  
    return x * y;  
}
```



```
foo (int x, y) {  
    return x * y;  
}
```

Background – Why, Why Not

- Why Not Inlining

- Increase in code size (obvious)
- Increase register pressure (more spills → performance degrades)
- Increase compilation time (for dynamic compilation)

A small experiment

```
class test {
    public static void main (String [] args) {
        int x = 0; int i = 0; int y = 0; int z = 0;
        java.util.Random generator = new java.util.Random();
        long start = System.currentTimeMillis();
        while (x < 1000000000) {
            y = x+1; z = x-1;
            //i = assign(x, y, z);
            i = x * 2 - y + z; //manual inlining
            x++; }
        long stop = System.currentTimeMillis();
        System.out.println("Elapsed time: " + (stop - start));
    }
    //public static int assign (int x, int y, int z) {
    //    return 2 * x - y + z;}
}
```

With manual inlining:

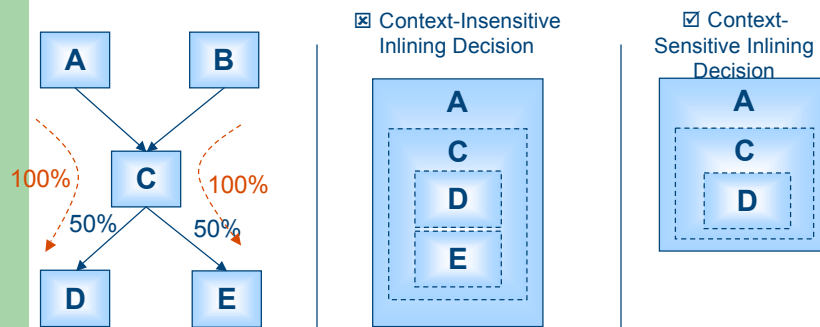
~ 5.8 ms

With no inlining:

~ 9 ms

Motivation: When to inline?

- Use profile information to find out which callers, callees, callsites are “hot”
- Without context, only one edge is evaluated to compute hotness → Context sensitivity is a must

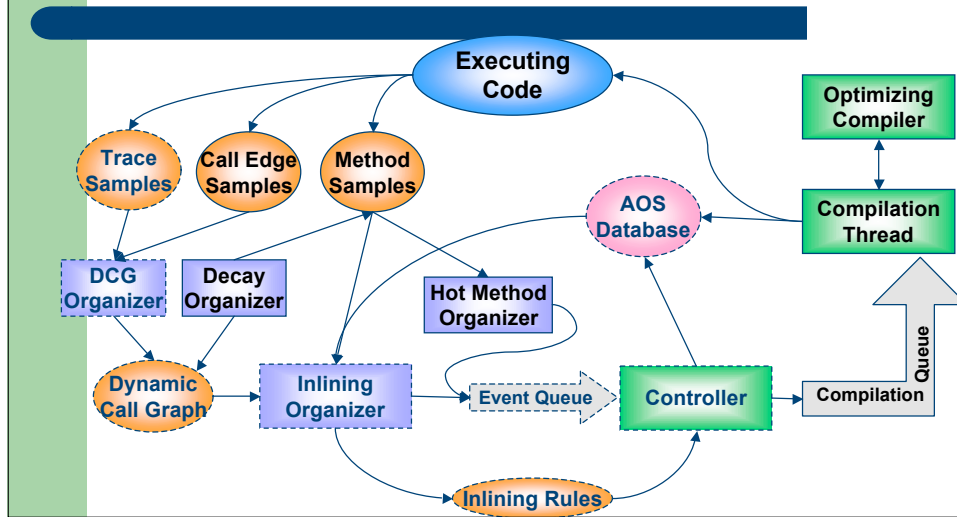


Problem (cont.)

- Offline or Online?
 - Offline: more thorough analysis, does not add to applications' run time
 - Online: applications are benefited *immediately* when running in VM's
- This paper is about context-sensitive online inlining

More information on Offline Context-Sensitive Inlining: "The Use of Traces for Inlining in Java Programs" by Bradel and Abdelrahman, LCPC04

Model overview: Jikes RVM

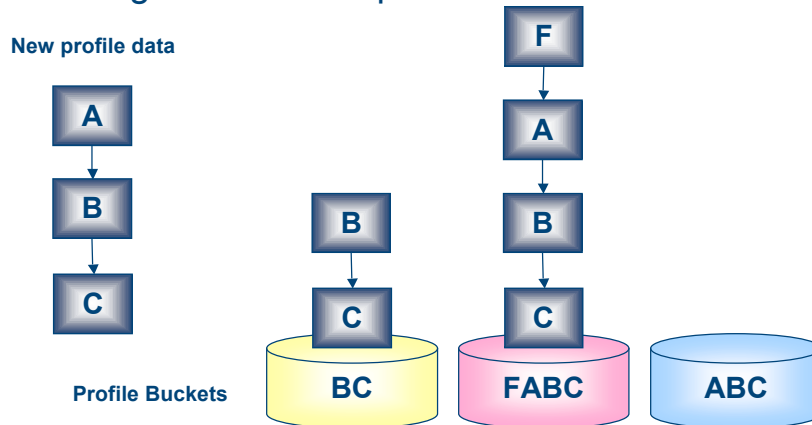


Implementation: Trace listeners

- Edge listeners (already in Jikes) collect $\langle \text{caller}, \text{callsite}, \text{callee} \rangle$
- Trace listeners collect the sequence of calls that end up at current callsite $\langle \text{caller}_1, \text{callsite}_1, \dots, \text{caller}_n, \text{callsite}_n, \text{callee} \rangle$
- How to deal with traces that partially match? ($\langle \text{BCD} \rangle$, $\langle \text{ABCD} \rangle \dots$)

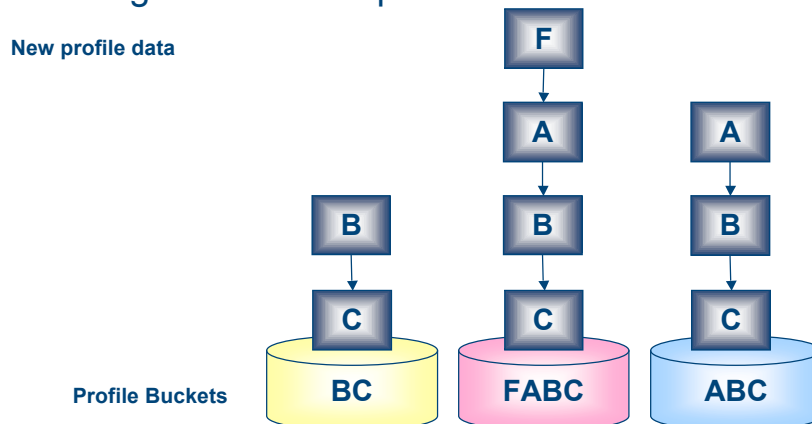
Implementation: Partial Context Matches

- During collection: separate traces



Implementation: Partial Context Matches

- During collection: separate traces



Implementation: Partial Context Matches

- During matching: allow partial matches

- Rules:

Compilation context:

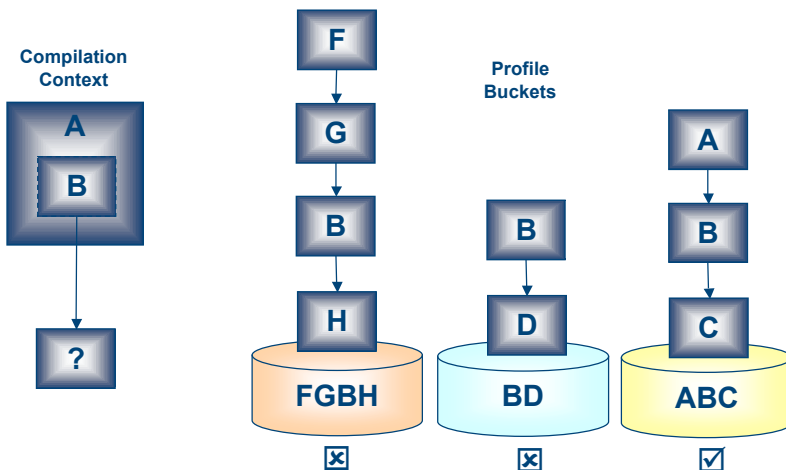
$\langle caller_{ck}, callsite_{ck}, \dots, caller_{c1}, callsite_{c1} \rangle$

All inlining rules with context

$\langle caller_{tj}, callsite_{tj}, \dots, caller_{t1}, callsite_{t1} \rangle$ are applicable if

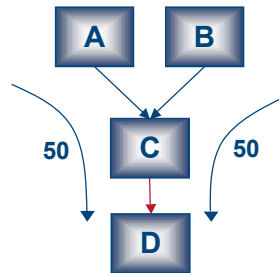
$$\begin{cases} caller_{ci} = caller_{ti} \\ callsite_{ci} = callsite_{ti} \end{cases} \forall i : 1 \leq i \leq \min(k, j)$$

Implementation: Partial Context Matches



Profiling Policies – how sensitive?

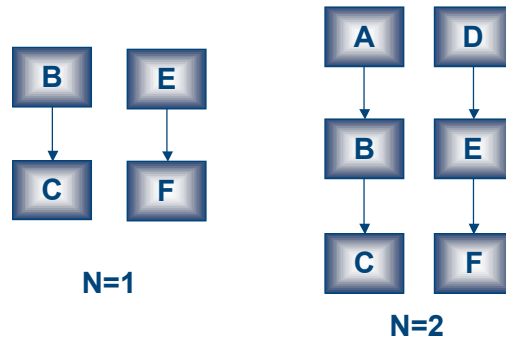
- Too much sensitivity :
 - High overhead (collecting too much trace information)
 - Profile diluting: delayed optimization



Profiling Policies

- Ideal sensitivity: not realistic
- Fixed-level sensitivity:
 - With no context: level = 1
 - With context: level = 2 → n
- Adaptive Sensitivity
 - Parameterless calls
 - Class methods
 - Large methods
 - Hybrid

Fixed-level Sensitivity



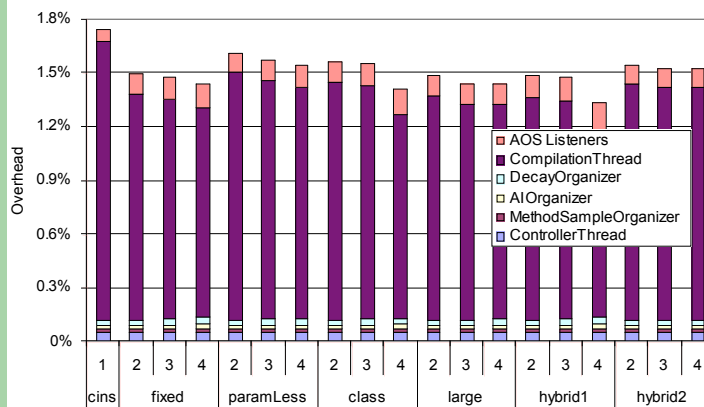
Adaptive sensitivity

- Parameterless:
 - Idea: Context is less important if no information is being passed from caller to callee (some exceptions)
- Class methods:
 - Why?
- Large methods: large methods should never be inlined.
- Hybrid

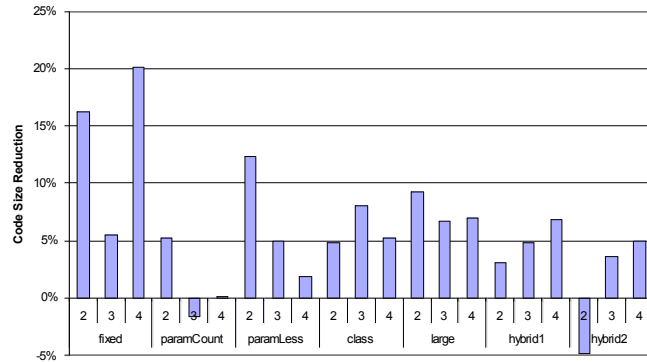
Evaluation

- Jikes RVM on Redhat Linux, Pentium 3 with 2.5 GB of RAM.
- FastAdaptiveSemispace configuration
- SPECjvm98 and SPECjbb2000
- For SPECjvm98, best run of 20 is reported (?)
- Most notable results are reductions in code size

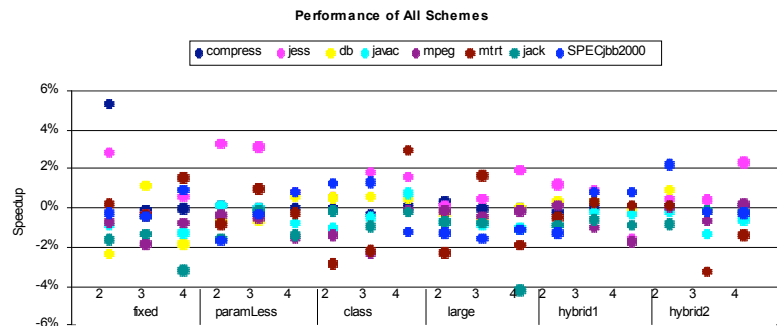
Results - Overhead



Result – Code size



Results - Performance



Critical Evaluation

- Good:
 - Online algorithm, programs can benefit from this while running.
 - Trade 10-20% code size for -1% performance
 - Evaluation is done on a real virtual machine, not simulation
- Bad:
 - Performance is rather disappointing (since performance is a more important bottleneck of java applications – rather than code size)
 - Heuristics for context sensitivity are too simple (a complicated method is proposed but was not evaluated)

Critical Evaluation (cont.)

Thoughts:

- The effect of cache misses/hits on Inlining?
Is it possible that for some reasons, cache misses increase while inlining, which is holding the performance back?
- How about partial inlining using context information?

Related Work

- Using Traces Information to Inline (Bradel, LCPC04)
 - Traces: sequence of basic blocks
 - Offline: need to collect traces information
 - Inline traces that are hot
 - Leads to substantial code size increase (47%)
 - Gains 10% on execution time
 - Compilation time is increased 29%

Related Work

- Automatic Tuning of Inlining Hueristics (Cavazos, SC05)
 - Use Genetic Algorithm (“natural” selection)
 - Start with several *individuals*, characterized by several values :
 - *CALLEE_MAX_SIZE*
 - *ALWAYS_INLINE_SIZE*
 - *MAX_INLINE_DEPTH*
 - *CALLER_MAX_SIZE*
 - *HOT_CALLEE_MAX_SIZE*
 - Each individual is evaluated using a *fitness* function
 - The one that has the highest fitness *survives*
 - Results: 17% execution time reduction on Intel platform, 6% on PowerPC, no code size change reported

Cavazos, SC05 (cont.)

```
inliningHeuristic(calleeSize, inlineDepth, callerSize) {
    if calleeSize > CALLEE_MAX_SIZE return NO;
    if calleeSize < ALWAYS_INLINE_SIZE return YES;
    if inlineDepth > MAX_INLINE_DEPTH return NO;
    if callerSize > CALLER_MAX_SIZE return NO;
    return YES;
}

inlineHotCallSite (calleeSize) {
    if calleeSize > HOT_CALLEE_MAX_SIZE return YES;
}
```