Dynamic Analysis for FDO of OOPLs

Arnold & Ryder, PLDI'01
Arnold, Hind, Ryder, OOPSLA'02

- What is FDO?
- An effective sampling profiling framework
- How to validate experimentally the cost (overhead incurred) and precision?
- Adaptive optimization with FDO
  - Optimizations used
  - How to measure performance gain?

Compilation Options

- Compiled method at first use with fixed set of optimizations (Just In Time or JIT compiler)
- Selective optimization of *hot methods* through compilation
- *Feedback directed optimization* (FDO) for longer-running applications
  - Profiling used to choose *what* and *how* to optimize
  - Offline profiles used since online profile collection often degraded performance due to cost of code instrumentation

☹ Translation incurs runtime overhead

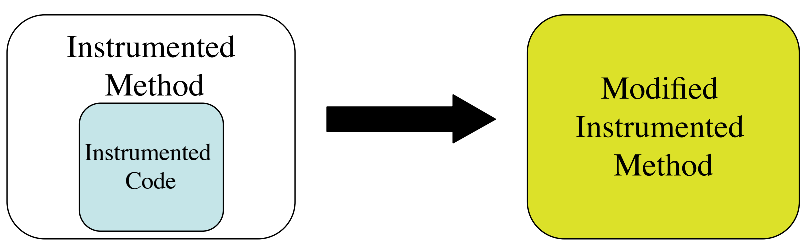
☺ Allows compiler to make judgments using runtime information

Problems with Online FDO

- What is *instrumentation*?
  - e.g., recording object field accesses
- Instrumentation overhead
  - Profiling interval must be short, but then may not be representative
  - Need a way to stop instrumented execution
    - Dynamic instrumentation
- Our contribution:

  General framework for instrumentation sampling and experiments with it.

Our Contribution



High Overhead ***Low Overhead***

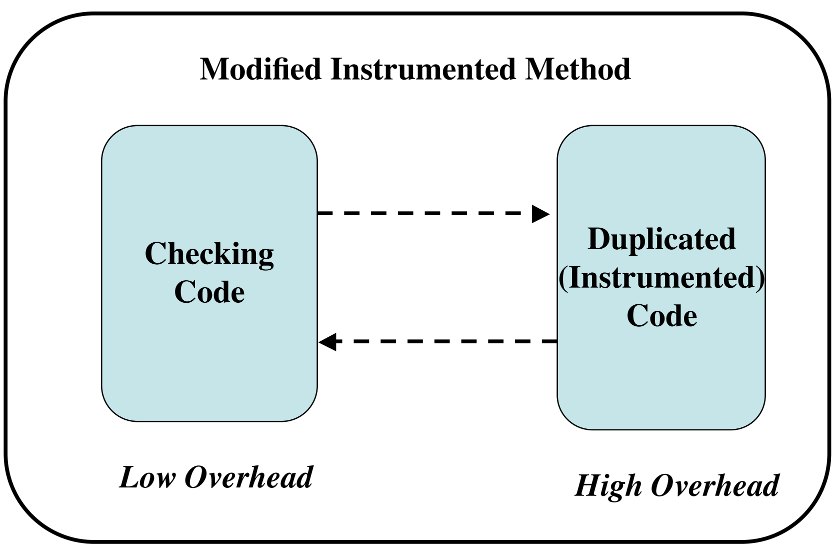
Achieved through our new sampling framework, independent of architecture or operating system.

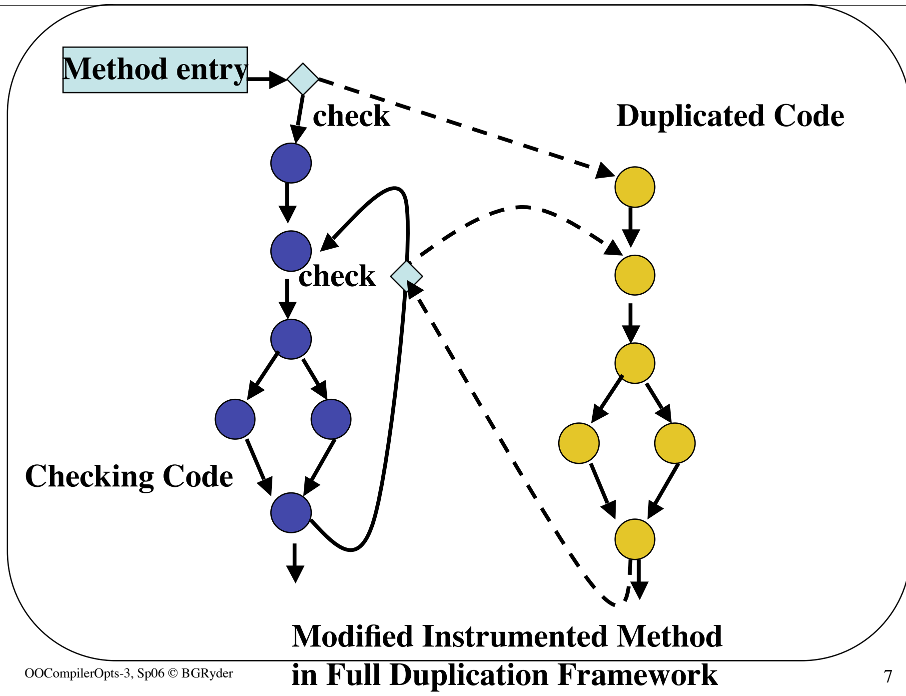
Advantages

In our low overhead sampling framework

- **Instrumentation can be run longer for greater accuracy**
- **Can apply multiple instrumentations at same time without framework modification;**
- **Most instrumentation incorporated without modification**
- **Framework is *tunable* allowing tradeoffs between overhead and accuracy (i.e., adjustable sampling rates)**
- ***Deterministic sampling* simplifies debugging**

Our Framework





Potential Disadvantages

- Code space may be doubled
  - VM will apply instrumentation selectively
    - Only in frequently executing methods
  - Other space-saving versions of framework
  - Empirical results show space usage is not a problem
- Sampled profile not same as exhaustive profile
  - Can't determine that an event did NOT occur
  - Can't check "for every iteration" assertions

Full-Duplication Framework

- **Key Property**
  - ***The number of checks executed in the checking code is less than or equal to the number of back edges and method entries executed, independent of the instrumentation being performed***

Counter-based Sampling

- **Take a sample after executing *n* checks**
- **Each check is:**

```
globalCounter --;
If (globalCounter ==0) {
  takeSample();
  globalCounter = resetValue;
}
```

- **Advantages**
  - **Simple, but effective**
  - **Hardware independent**
  - **Tunable, flexible sampling rate**
  - **Can be used with any VM**

Framework Measurment

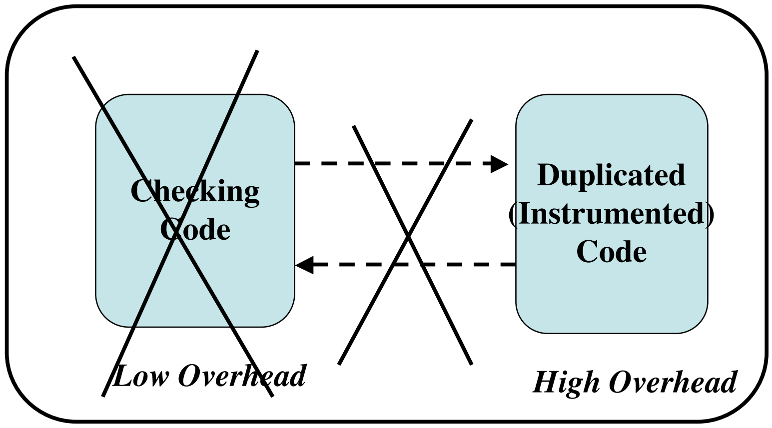
- **Implemented in IBM's Jalapeno JVM**
- **10 benchmarks**
  - ***SPECjvm98*(input size 10), *Volano, pBob, opt-compiler***
  - **Running times from 1.1-4.8 seconds**
  - **Class file sizes from 10K-1,517K bytes**
  - **Machine 333Mz IBM RS/6000 powerPC 604e with 2096Mb RAM running AIX 4.3**
- **Instrumented all methods in applications and libraries**

Instrumentation

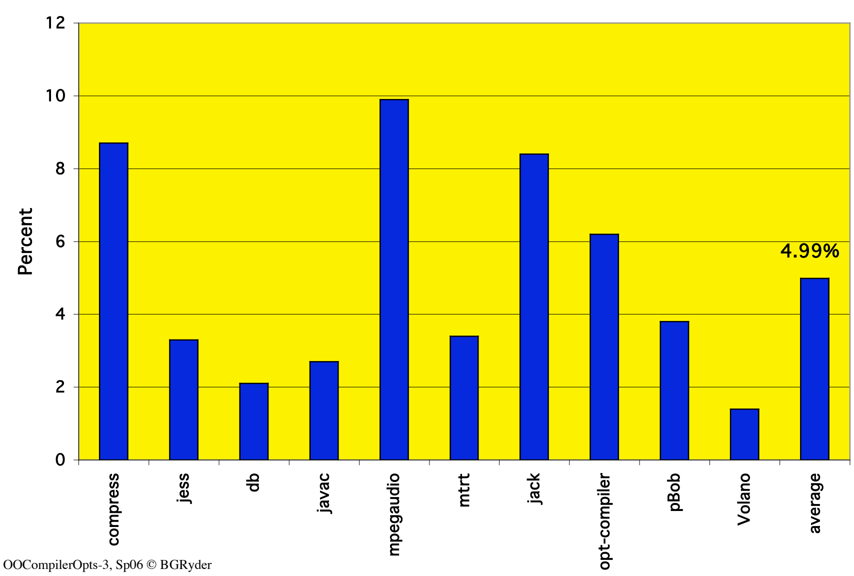
- **Call-edge:**
  - **Collect caller, callee, call-site within caller at method entry**
  - **One counter per call edge**
- **Field-access:**
  - **One counter per field of each class**
  - **Each *putfield, getfield* access instrumented**

Exhaustive Instrumentation Overhead

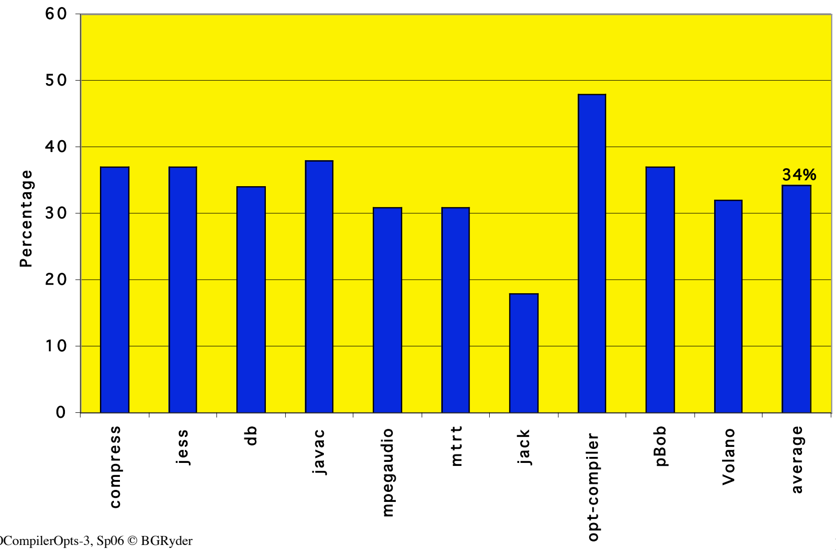
On average, 88% call-edge and 60% field-access



Time Overhead(Full-Dup)



Compile-time Increase (Full-Dup)



Sampling Cost (Full Dup)

Sample Interval	Overhead (Full-Dup)	Call-edge Accuracy	Field-access Accuracy
1	182%		
10	29%		
100	10%		
1,000	6%		
10,000	5%		
100,000	5%		

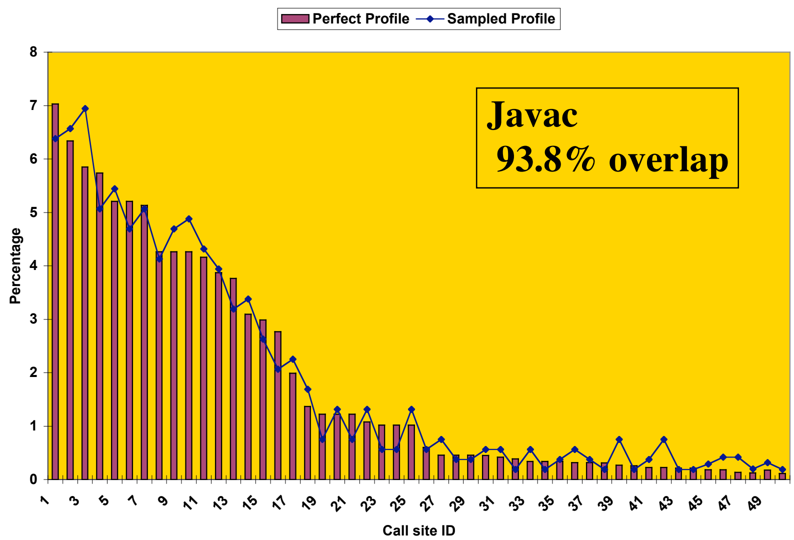
Measuring Precision

- **Run sampling framework to record call edges**
- **Run *perfect profile* recording every call**
- **Compare percentage of sample collected attributed to a particular call edge to corresponding percentage in the perfect profile.**

Measuring Accuracy

- ***Overlap* is minimum of these two percentages**
- ***Overlap percentage* is sum of overlaps for all edges (Feller 98)**
  - **Any sample will be less than or equal to 100%**
  - **A sample identical to perfect profile has 100% overlap**
  - **If sampling overestimates the percentage for some call site then it must underestimate the percentage for another call site**

Sample & Perfect Profiles (Javac)



Sampling Cost + Accuracy
(Full Dup)

Sample Interval	Overhead (Full-Dup)	Call-edge Accuracy	Field-access Accuracy
1	182%	100%	100%
10	29%	99%	100%
100	10%	98%	99%
1,000	**6%**	**94%**	**97%**
10,000	**5%**	**82%**	**94%**
100,000	5%	71%	83%

FDO Adaptive Optimization

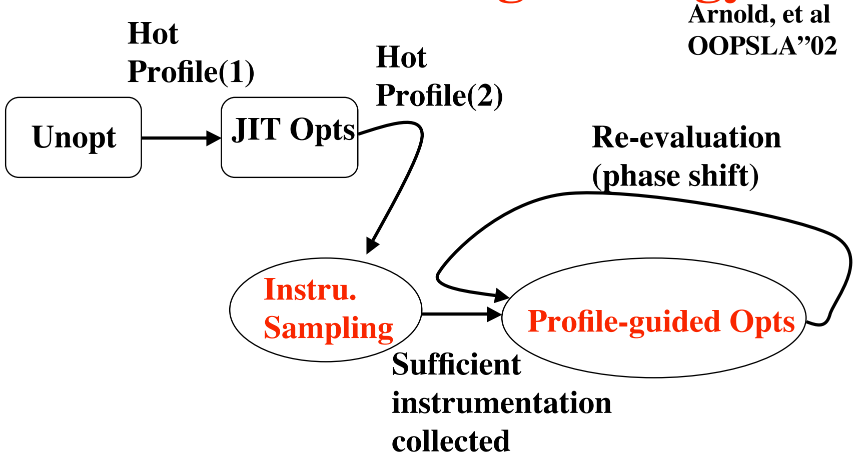
- In Jalapeno,
  - Compile application and run
  - Identify "hot" methods and insert instrumentation
  - Collect sampling instrumentation
  - Recompile the "hot" methods with feedback-directed optimizations using sampling information
  - Optionally resume sampling as long as may want to recompile

Online FDO Experiments

- Embed full duplication framework in Jikes Research VM for adaptive optimization trials
  - Insert instrumentation at highest optimization level (O2) so see optimization effects in profile
  - Instrumentation is intraprocedural edge counters
  - Optimizations used
    - Splitting
    - Code positioning (to increase code locality)
    - Loop unrolling
    - Adaptive inlining

Online Profiling Strategy

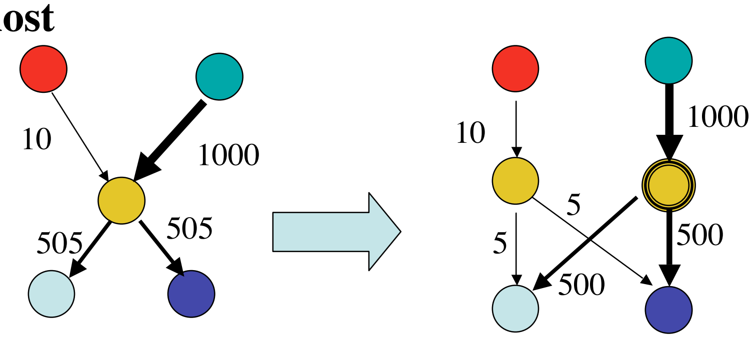
Arnold, et al
OOPSLA"02



Arrows represent recompilation steps in the 2-phased profiling

Splitting

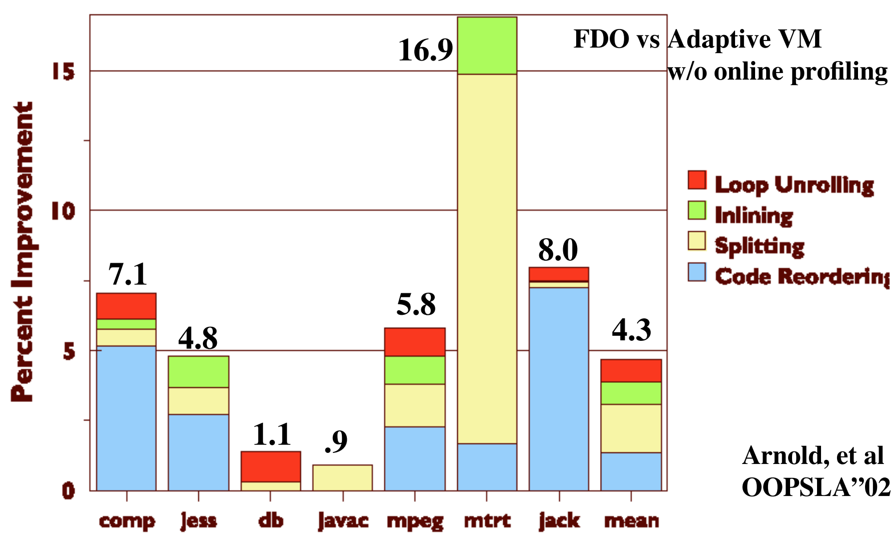
- **Splitting is tail duplication of code to eliminate merges that cause dataflow info to be lost**



How to measure performance?

- Factors
  - Overhead of instrumentation
  - Effectiveness of FDO's
  - Underlying adaptive optimization system
- Measure steady-state performance of SpecJvm98 codes
  - Requires running benchmarks in harness multiple times (to total time of 4 minutes on size 100)

Peak Performance Gains



Arnold, et al
OOPSLA"02

Arnold OOPSLA'02

SPECjbb2000

Arnold, et al
OOPSLA"02

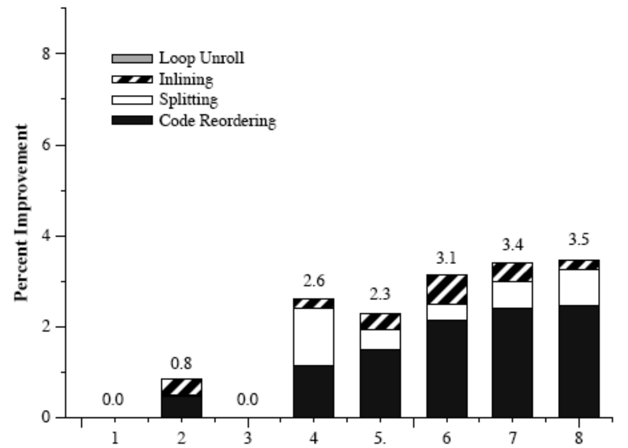
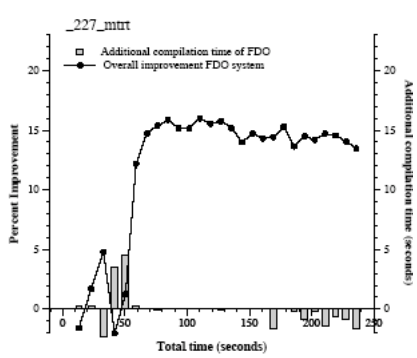
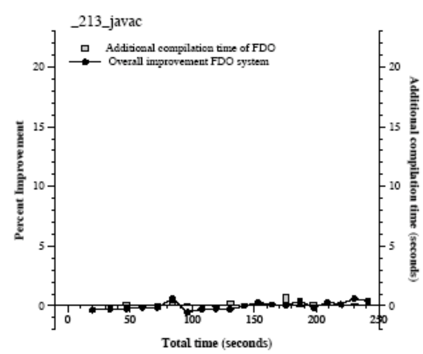


Figure 10: Performance improvement of FDO on the SPECjbb2000 server benchmark

Comparison

Arnold, et al
OOPSLA"02





Mtrt - successful FDO

Javac, unsuccessful FDO

Specjvm98

← FDO compilation stats →

Benchmarks	Application Characteristics		Compilation Statistics (with FDO)						Space Overhead
			Total #	Percent Breakdown					
	# Runs	Best time	compilations	Base	O0	O1	O2	INST/FDO	% Increase
compress	11	18.8	382	93	2	3	1	1	6.3
jess	36	6.3	915	84	6	6	2	1	6.2
db	14	17.3	399	94	2	2	1	1	5.8
javac	20	10.9	1,575	70	16	32	1	1	4.6
mpegaudio	11	19.9	704	75	11	10	3	2	6.9
mtrt	54	4.1	634	78	8	10	2	1	6.6
jack	15	15.5	738	80	10	6	3	1	6.5
Geomean	19	11.5	787	82	6	7	2	1	6.0

Table 2: Recompilation statistics and space overhead for the SPECjvm98 benchmarks

Arnold, et al
OOPSLA"02