Performance Potential of Optimization Phase Selection During Dynamic JIT Compilation

Michael R. Jantz Prasad A. Kulkarni

Electrical Engineering and Computer Science, University of Kansas
{mjantz,kulkarni}@ittc.ku.edu

March 17, 2013

Phase Selection in Dynamic Compilers

- Phase Selection – customizing set of optimizations applied for each method / program to generate the best quality code
- Static solutions do not necessarily apply to JITs
- Heuristics improve *startup* performance.
- Is it possible for phase customization to improve peak throughput (aka *steady-state* performance)?

Our Goals

- Understand optimization behavior relevant to phase selection
- Quantify the performance potential of customizing phase selections in online JIT compilers
- Determine if *current* state-of-the-art heuristics achieve ideal performance

Our Experimental Framework

- Uses the server compiler in Sun/Oracle's HotSpot™ JVM
  - Applies a fixed optimization set to each compiled method
  - Imposes a strict optimization ordering
  - Modified to optionally enable / disable 28 optimizations
- Two benchmark suites with two inputs:
  - SPECjvm98 (input sizes 10 and 100)
  - DaCapo (small and default)
- Phase selection applied to one *focus method* at a time
  - Profiling run to determine hottest methods
  - Select methods that comprise at least 10% of total program runtime (53 focus methods across all of the benchmarks)

Performance Measurement

- All experiments measure *steady-state* performance
  - Hot methods pre-compiled
  - No compilation during steady-state iterations
- Run on a cluster of server machines
  - CPU: four 2.8GHz Intel Xeon processors
  - Memory: 6GB DDR2 SDRAM, 4MB L2 Cache
  - OS: Red Hat Enterprise Linux 5.1

Analyzing Behavior of Compiler Optimizations

- In some situations, applying an optimization may have detrimental effects
- Optimization phases interact with each other
- Experiments to explore the effect of optimization phases on code quality

$$\frac{T(OPT < defOpt - x >) - T(OPT < defOpt >)}{T(OPT < defOpt >)} \quad (1)$$

Impact of each Optimization over Focus Methods

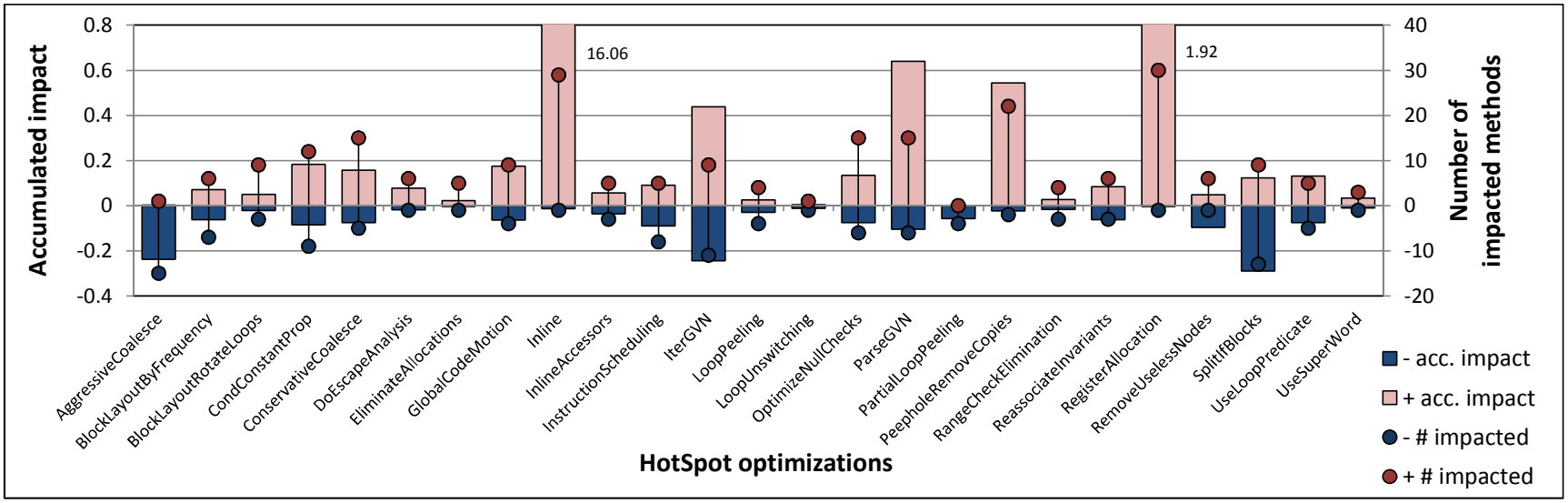

Figure 1: **Left Y-axis:** Accumulated positive and negative impact of each HotSpot optimization over our focus methods (non-scaled). **Right Y-axis:** Number of focus methods that are positively or negatively impacted by each HotSpot optimization.

- Optimizations not always beneficial to program performance
- Most optimizations have occasional negative effects

Impact of each Optimization over Focus Methods

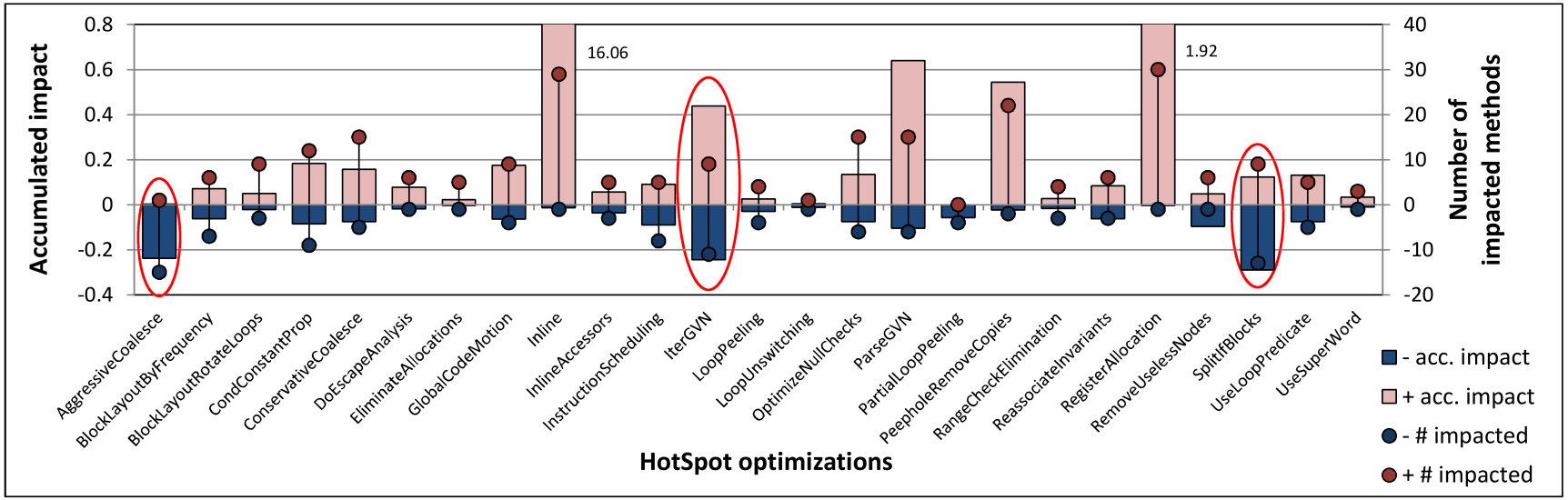

Figure 1: **Left Y-axis:** Accumulated positive and negative impact of each HotSpot optimization over our focus methods (non-scaled). **Right Y-axis:** Number of focus methods that are positively or negatively impacted by each HotSpot optimization.

- Some optimizations show degrading impact more often

Impact of each Optimization over Focus Methods

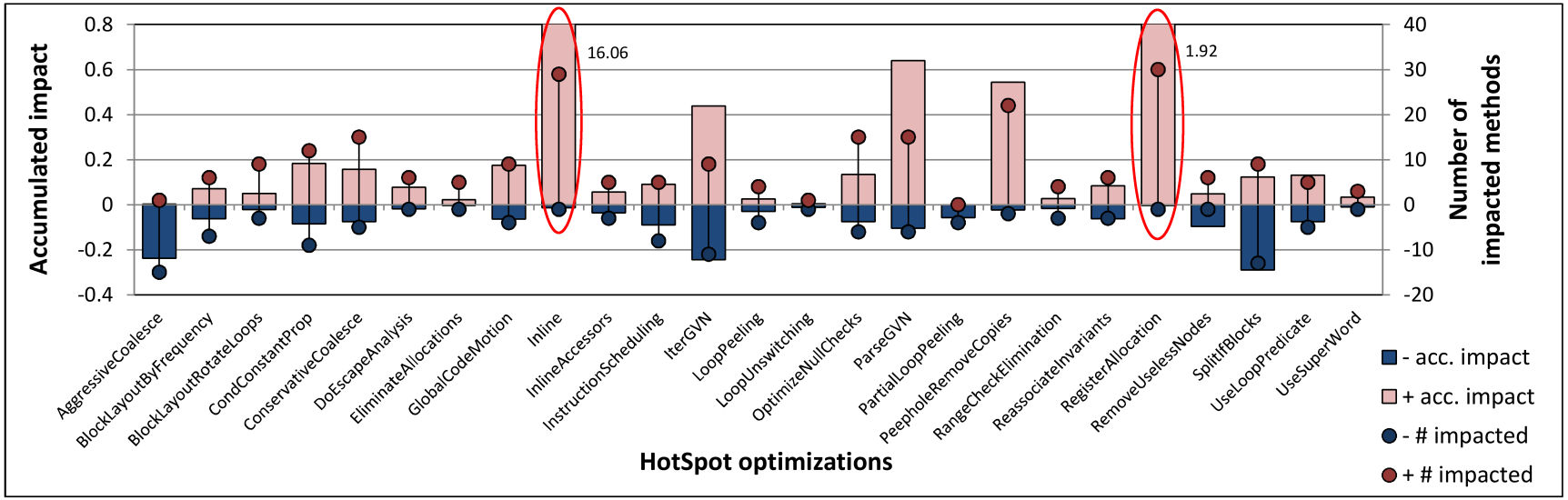

Figure 1: **Left Y-axis:** Accumulated positive and negative impact of each HotSpot optimization over our focus methods (non-scaled). **Right Y-axis:** Number of focus methods that are positively or negatively impacted by each HotSpot optimization.

- Most optimizations only have marginal impact on performance
- Most beneficial is method inlining, followed by register allocation

Impact of Optimizations for each Focus Method

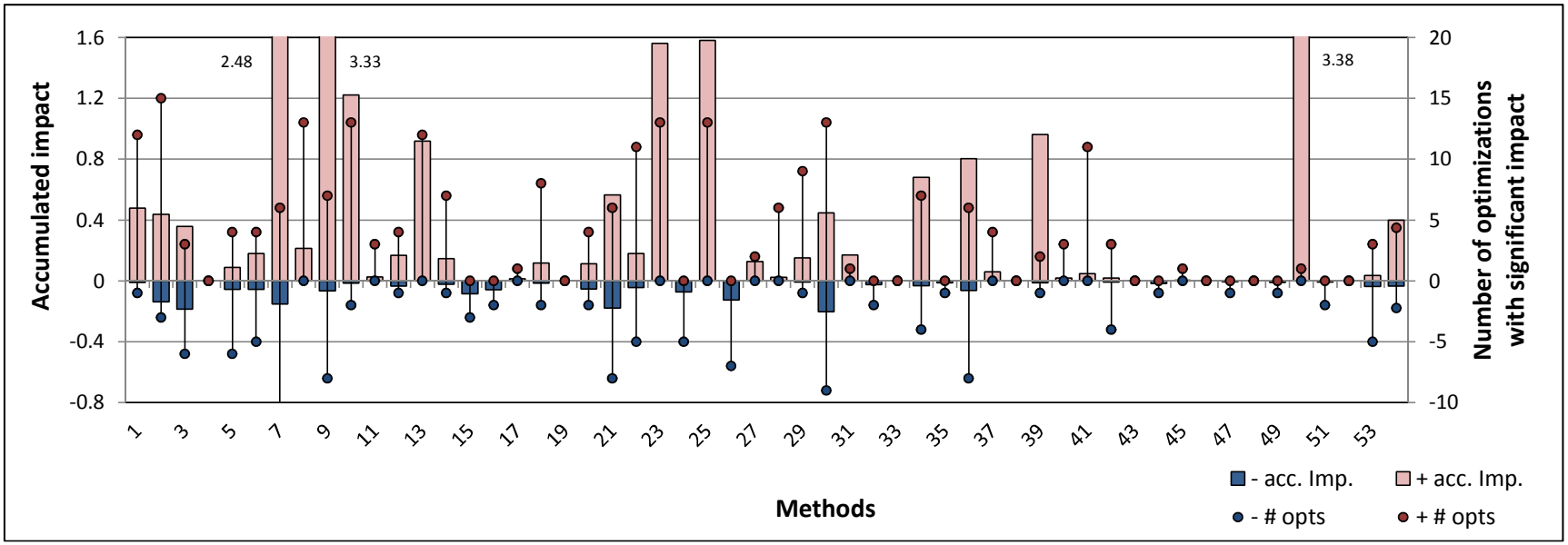

Figure 2: **Left Y-axis:** Accumulated positive and negative impact of the 25 HotSpot optimizations for each focus method (non-scaled). **Right Y-axis:** Number of optimizations that positively or negatively impact each focus method. The rightmost bar displays the average.

- Not many optimizations degrade performance (2.2, on average)
- Only a few optimizations benefit performance (4.4, on average)

Limits of Optimization Phase Selection

- Iterative search infeasible due to number of optimizations
- Employ long-running *genetic algorithms* (GA's) to find near-optimal phase selections
  - Evaluate group of phase selections in each GA generation
  - Random mutation and crossover to next generation's set of phase selections
  - 20 phase selections per generation over 100 generations

GA Results for Focus Methods

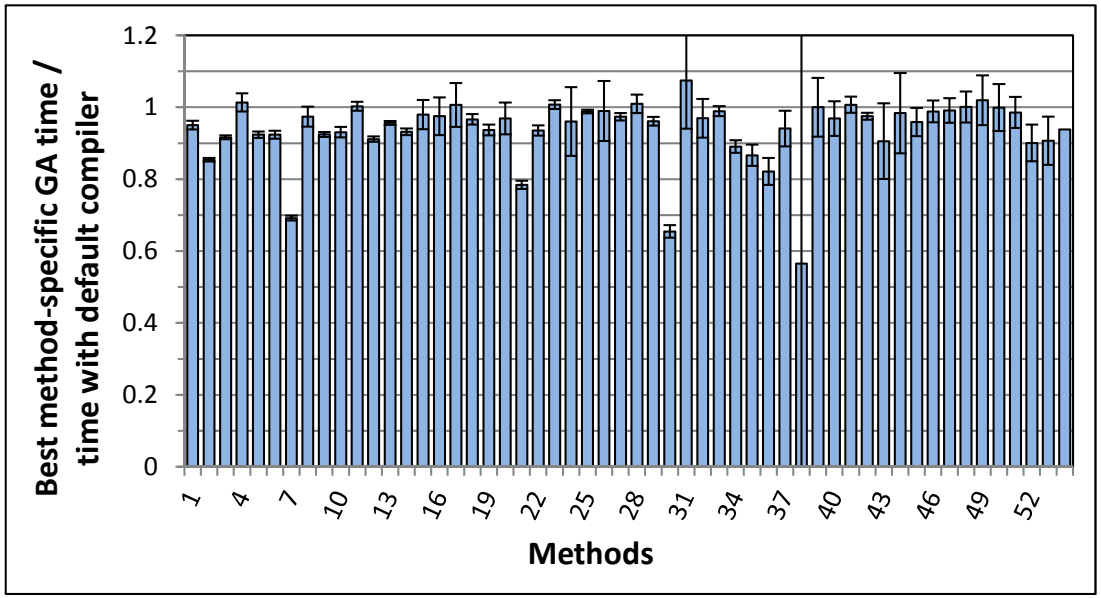

Figure 3: Performance of method-specific optimization selection after 100 GA generations. All results are scaled by the fraction of total program time spent in the focus method to show the runtime improvement for each individual method. The rightmost bar displays the average.

- Customizing optimization phase selections achieves significant gains
- Maximum improvement of 44%, average improvements of 6.2%

Effectiveness of Feature-Vector Based Heuristics

- Iterative searches not practical for JITs
- Previous works proposed using *feature-vector* based heuristics during online compilation [1, 2]
- GA results allow the first evaluation of these heuristics (compared to *ideal* phase selections)

Overview of Approach

- Training stage
  - Evaluate program performance for different phase selections
  - Construct a *feature set* to characterize methods
  - Correlate good phase selections with method features
- Deployment stage
  - Install learned statistical model into compiler
  - Extract each method's feature set at runtime
  - Predict a customized phase selection for each method

Our Experimental Setup

Scalar Features	Distribution Features	
Counters	**Types**	**ALU Operations**
Bytecodes	byte char	add sub
Arguments	int double	mul div
Temporaries	short long	rem neg
Nodes	float object	shift or
	address	and xor
		inc compare
Attributes	**Casting**	**Memory Operations**
Constructor	to byte	load load const
Final	to char	store new
Protected	to short	new array / multiarray
Public	to int	
Static	to long	**Control Flow**
Synchronized	to float	branch call
Exceptions	to double	jsr switch
Loops	to address	
	to object	**Miscellaneous**
	cast check	instance of throw
		array ops field ops
		synchronization

Table 1: List of method features used in our experiments

- Select method features relevant to HotSpot
- Use logistic regression to train our predictive model

Feature-Vector Based Heuristic Algorithm Results

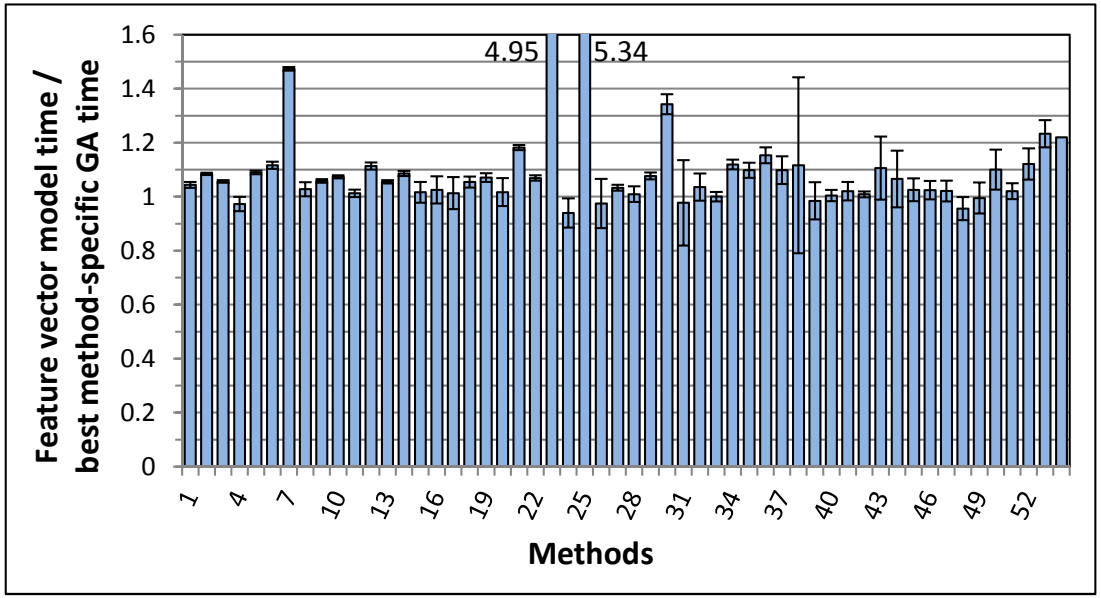

Figure 4: Effectiveness of method-specific feature-vector based heuristic. Training data for each method consists of all the other focus methods.

- On average, 22% worse than ideal, 14% worse than default compiler
- Feature-vector based heuristics cannot achieve improvements found by GA, but this result is similar to findings in previous research.

Feature-Vector Based Heuristic with Additional Analysis

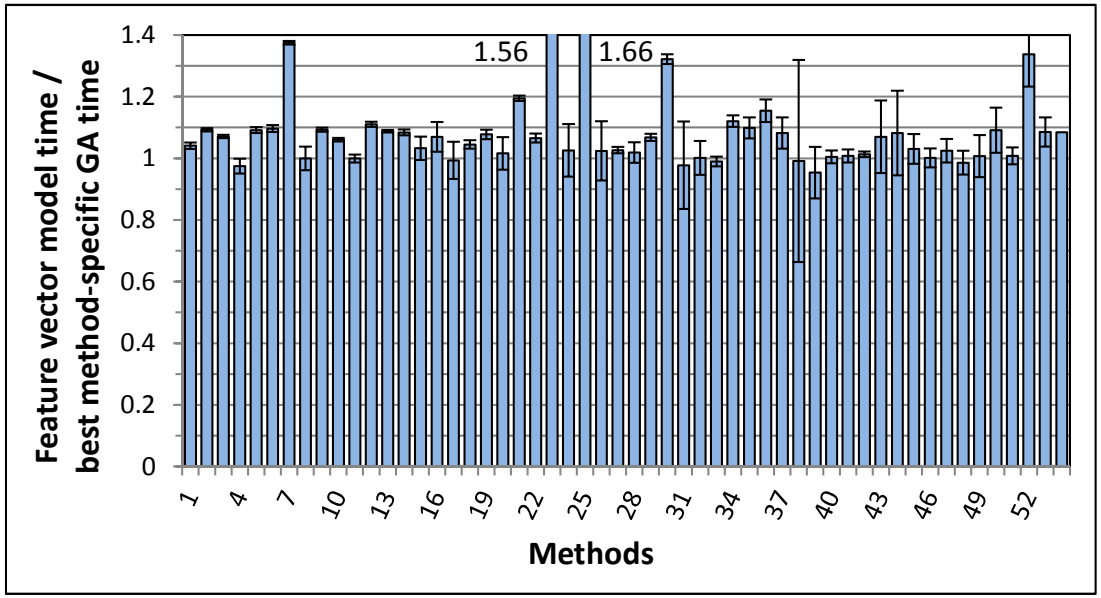

Figure 5: Experiments that use the observations from analysis of optimization behavior to improve the performance of feature-vector based heuristic algorithms for online phase selection

- Only predict ON/OFF setting for optimizations that show negative performance impact for at least 10% of methods
- Does not achieve ideal code quality, but only 8.4% worse than ideal

Conclusions

- Framework for optimization selection research in JITs
- Observations
  - Most optimizations have modest performance impact
  - Few optimizations are active for most methods
  - Most optimizations do not negatively affect performance
  - Modest potential for optimization phase selection
  - GA-based search yields 6.2% average improvement
  - Feature-vector based heuristics do not attain ideal performance
  - Suggested directions may improve phase selection heuristics

Questions

Thank you for your time. Questions?

References

[1] John Cavazos and Michael F. P. O'Boyle. Method-specific dynamic compilation using logistic regression. In *Proceedings of the conference on Object-oriented programming systems, languages, and applications*, pages 229–240, 2006.

[2] R.N. Sanchez, J.N. Amaral, D. Szafron, M. Pirvu, and M. Stoodley. Using machines to learn method-specific compilation strategies. In *Code Generation and Optimization*, pages 257 –266, April 2011.