The Use of Traces for Inlining in Java Programs

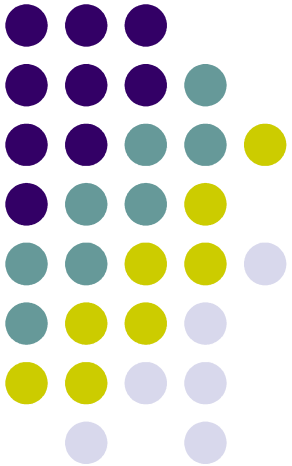
Borys J. Bradel

Tarek S. Abdelrahman

Edward S. Rogers Sr.Department of Electrical and Computer Engineering

University of Toronto

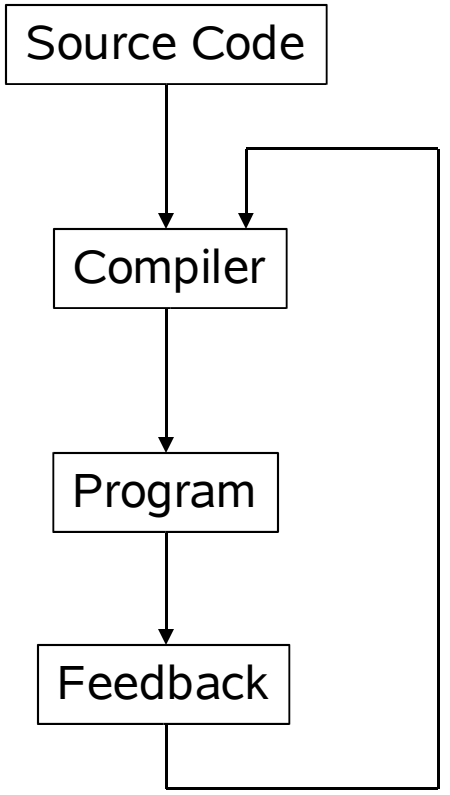
Toronto, Ontario, Canada



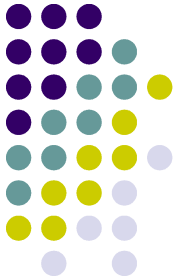
Introduction

- Feedback-directed systems provide information to a compiler regarding program behaviour

- Examples:
  - Jikes RVM [AFG+00]
  - Open Runtime Platform [Mic03]



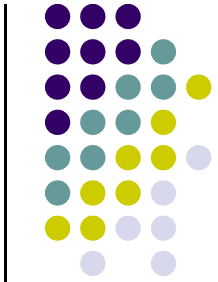
Work Overview



- Explore whether traces are useful in offline feedback directed systems

- Create trace collection system for Jikes
- Use traces to guide Jikes's built in optimizing compiler
  - Help with a single optimization, inlining
  - Improves execution time

Outline

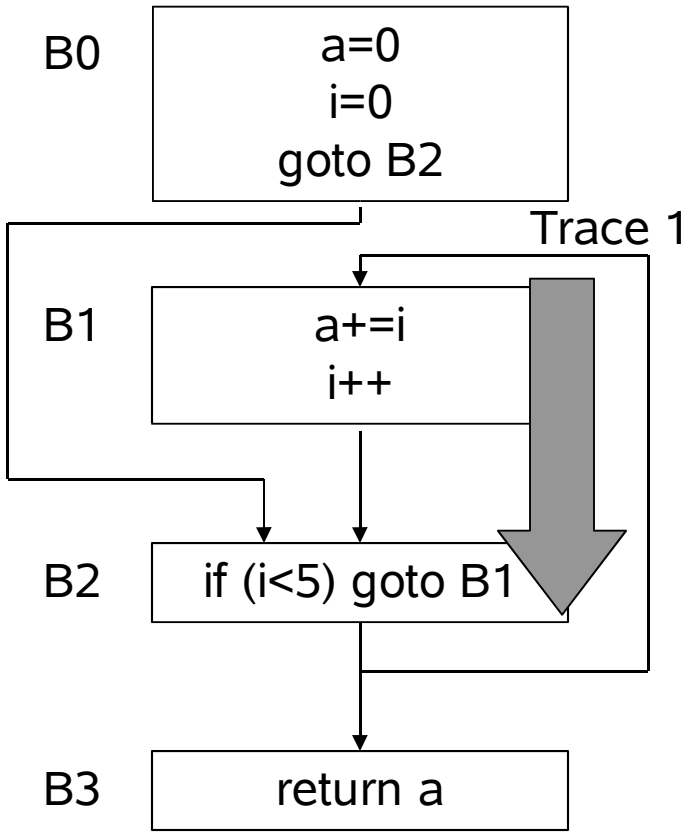


- Background
- Implementation
- Results
- Related work
- Conclusion

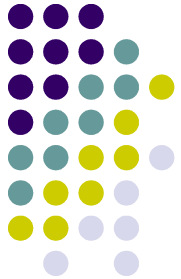
Trace Definition

- A trace is a frequently executed sequence of unique basic blocks or instructions

```
public static int foo() {
  int a=0;
  for (int i=0;i<5;i++)
    a++;
  return a;
}
```



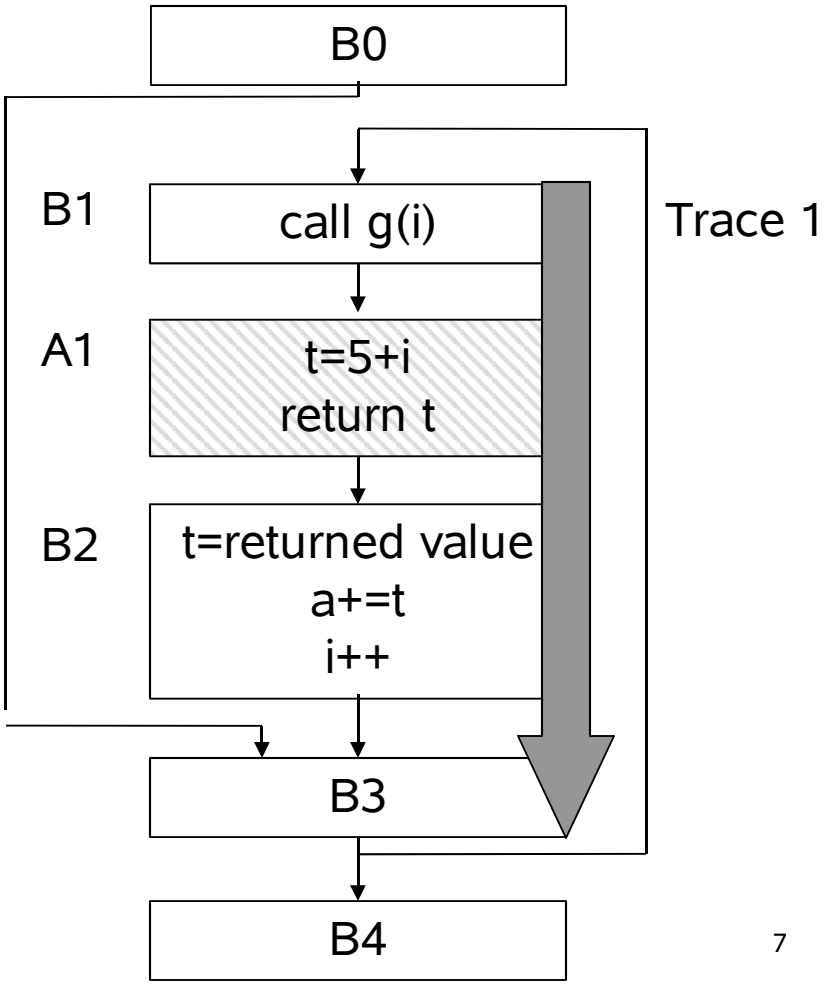
Traces and Optimization



- Traces may offer a better opportunity for optimization:
  - Enable inter-procedural analysis
  - Reduce the amount of instructions optimized
  - Simplify the control flow graph, allowing for more optimization

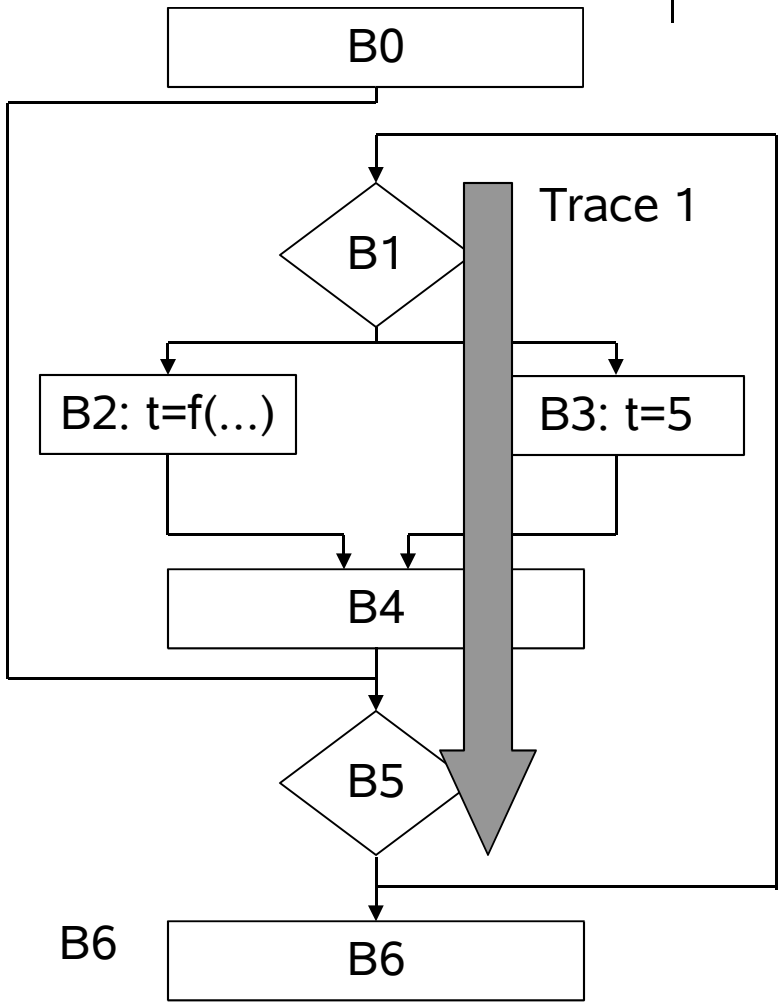
Multiple Methods

- Inter-procedural analysis without an additional framework
- Increase possibility of optimization
  - B1,A1,B2 can be simplified to two instructions
    - a+=(5+i)
    - i++



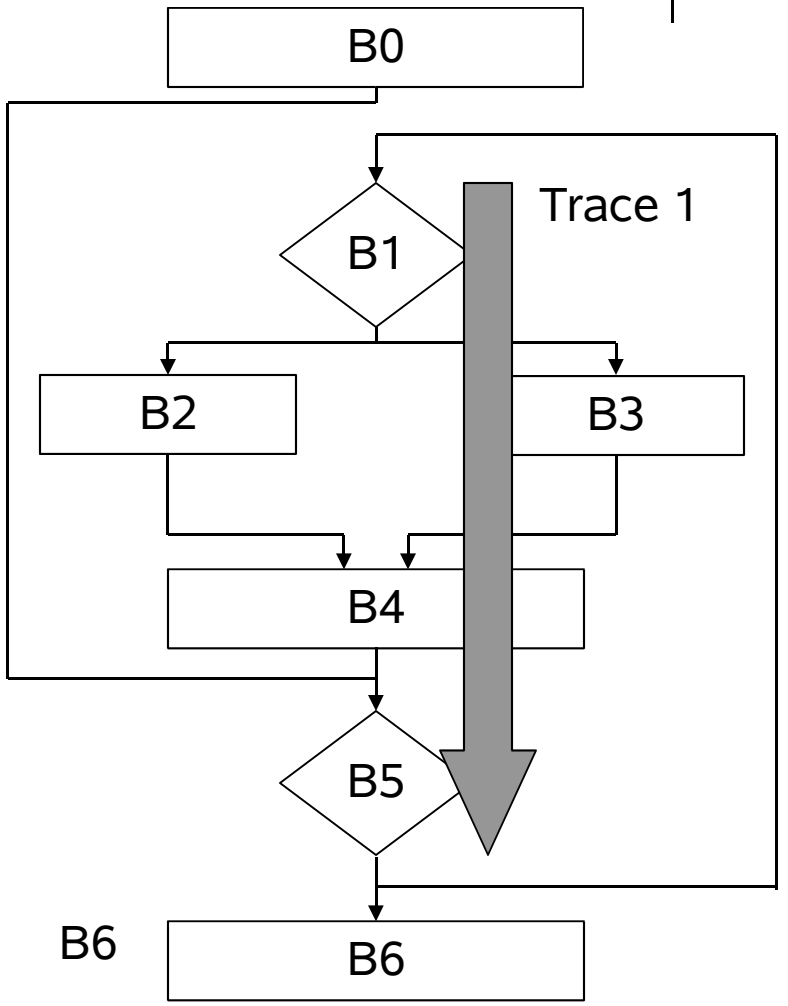
Fewer Instructions

- Fewer instructions to optimize
- May allow for extra optimization
  - If know that B3 is executed then know that t=5



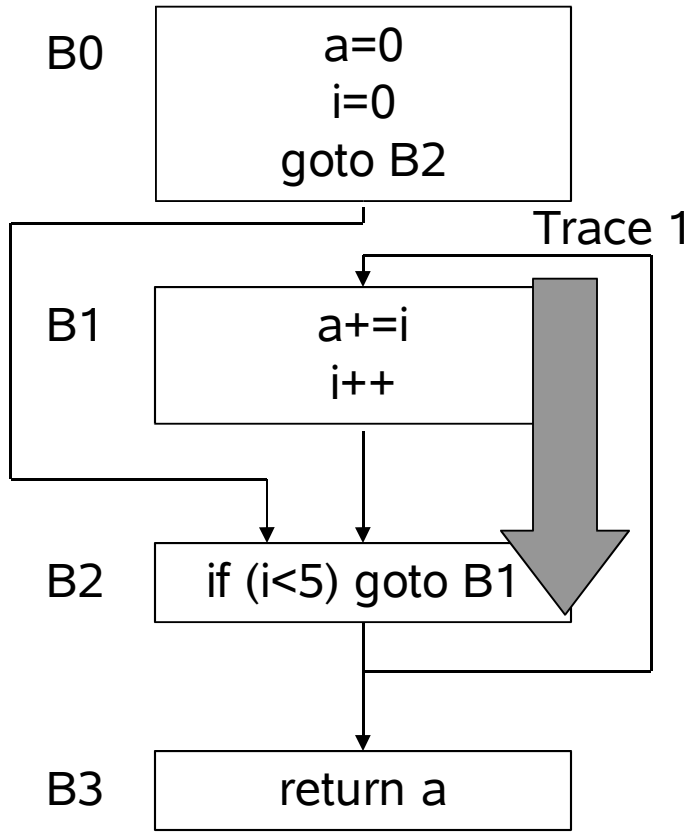
Trace Exits

- Traces usually contain many basic blocks
- Traces may not execute completely
  - Unlike basic blocks

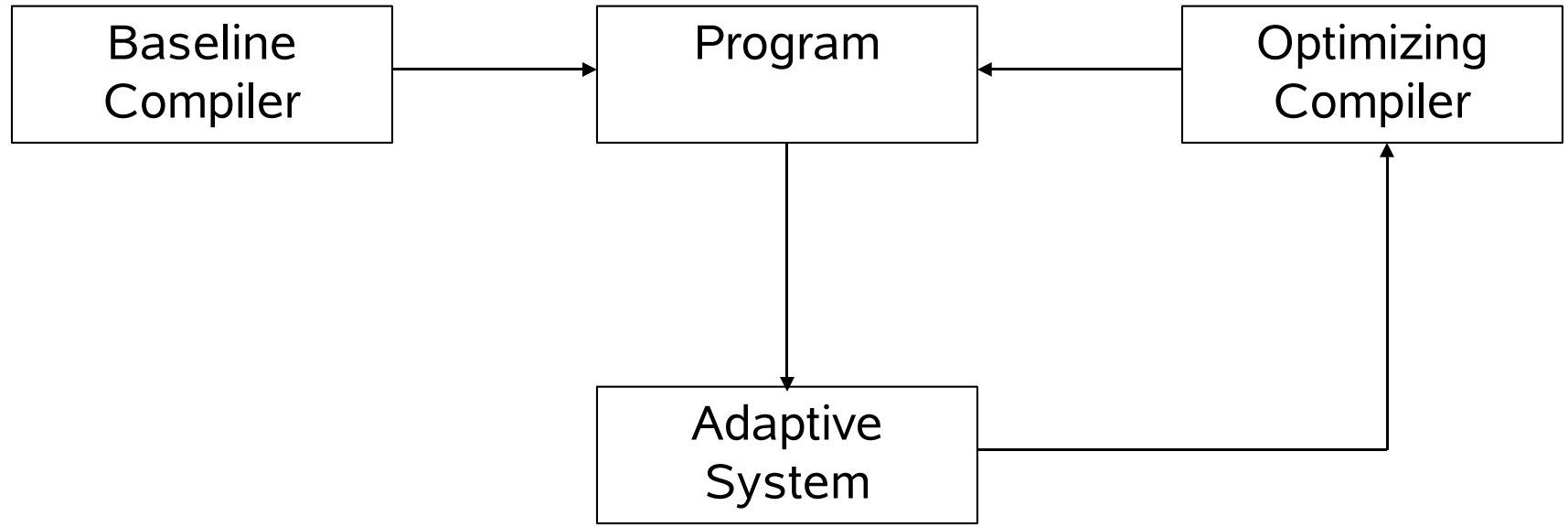


Trace Collection System

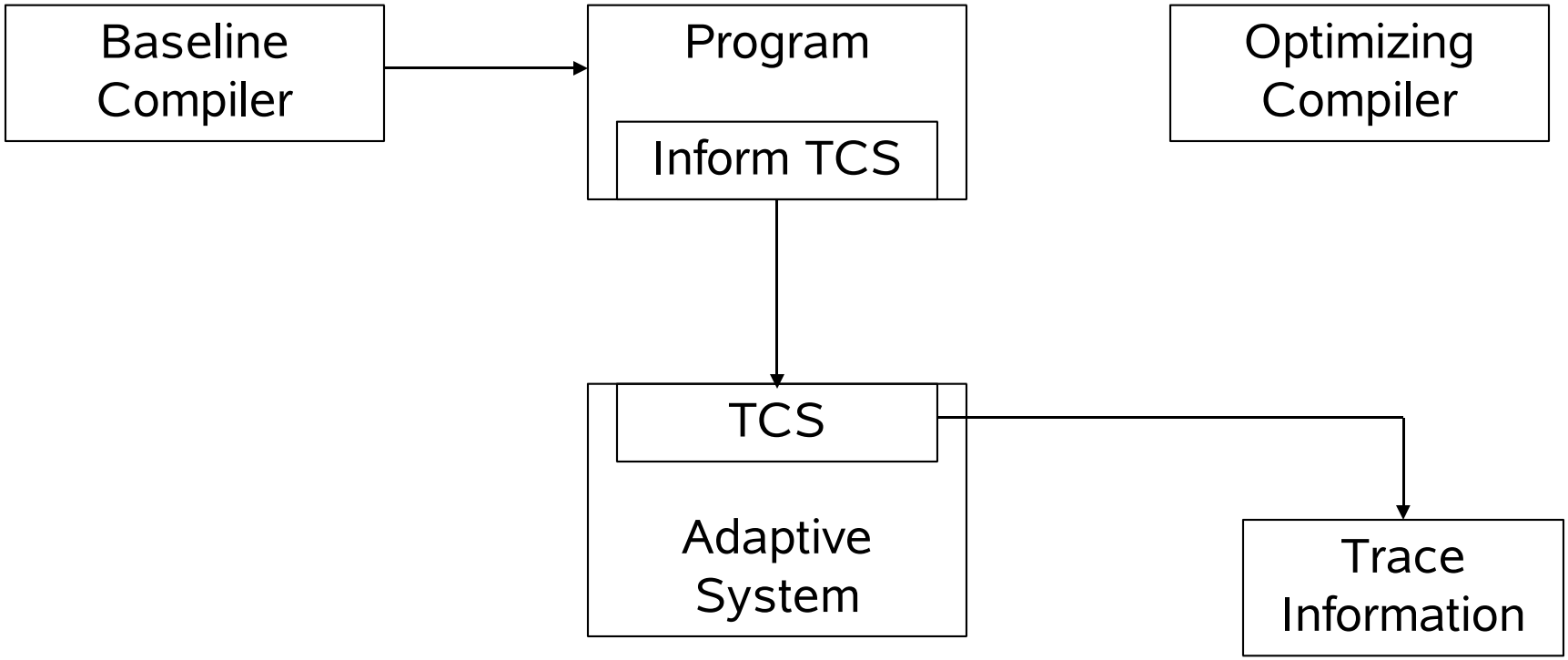
- Monitor program execution
- Record traces
- Start traces at frequently occurring events
  - Backward branches
  - Trace exits
  - Returns
- Stop at backward branches and trace starts
- Captures frequently executed loops and functions



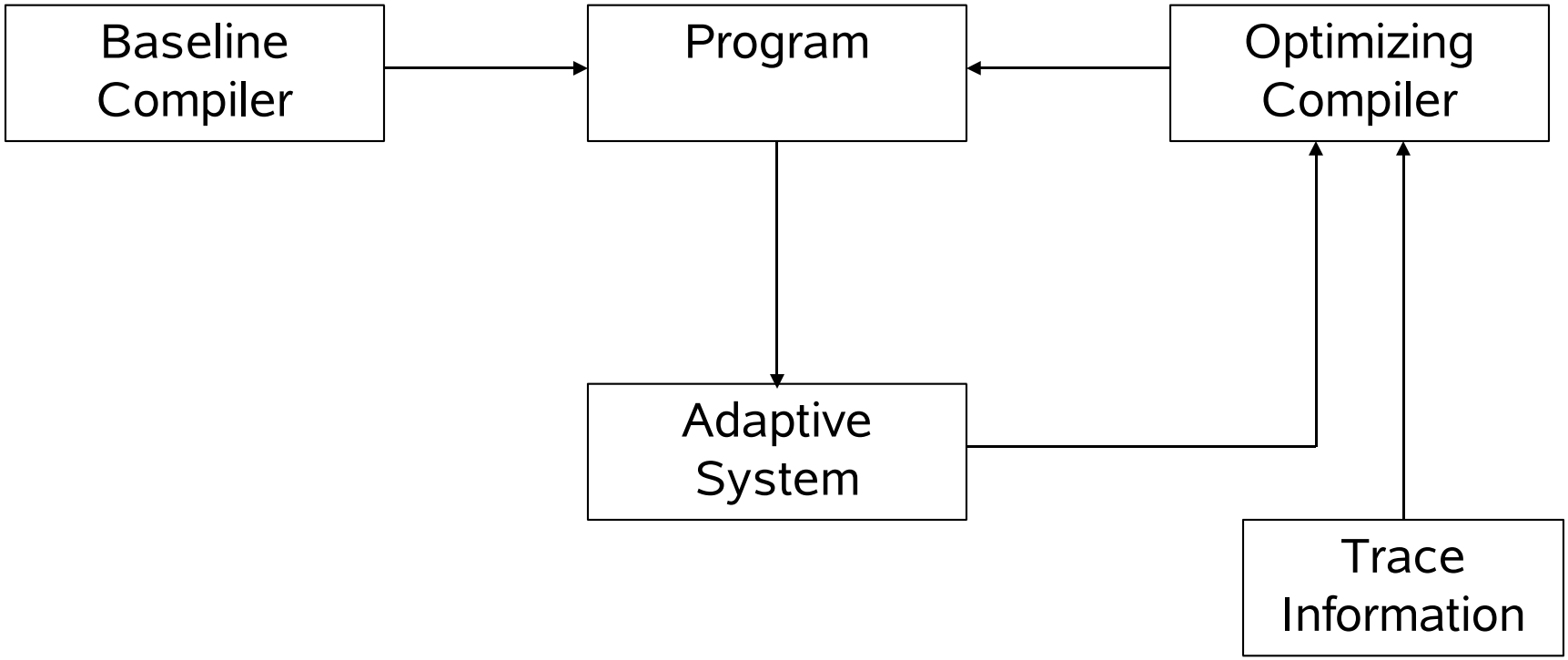
Jikes


Baseline Compiler
Program
Optimizing Compiler
Adaptive System

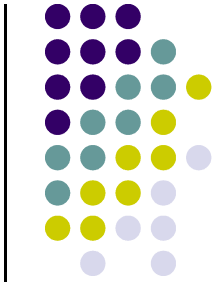
Jikes and our TCS



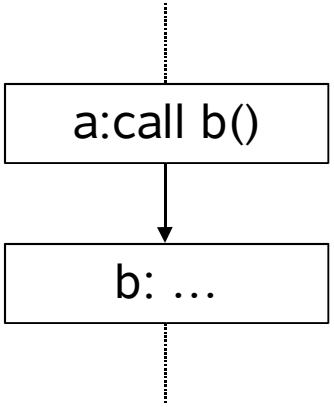
Jikes – Second Phase

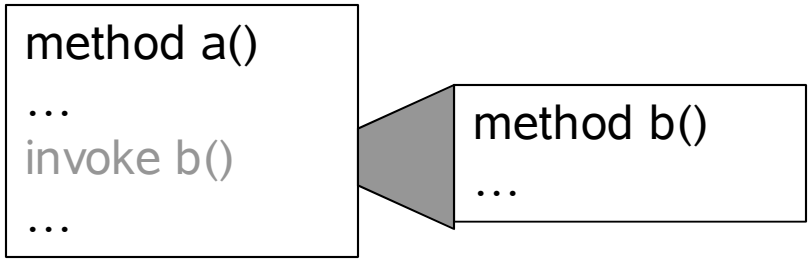


Inlining and Traces



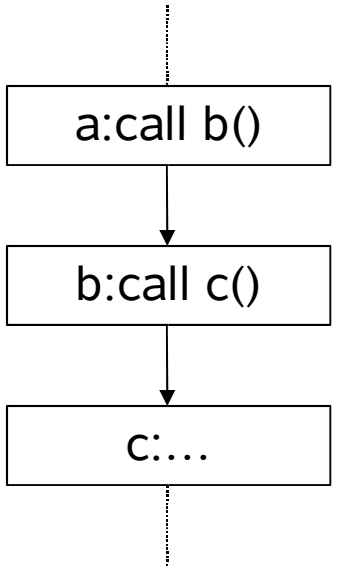
- Traces are executed frequently
- Therefore invocations on traces should be inlined
  - Reduce invocation overhead
  - Allow for more opportunities for optimization
- May lead to large code expansion



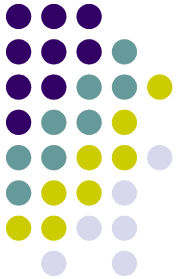


Code Expansion Control

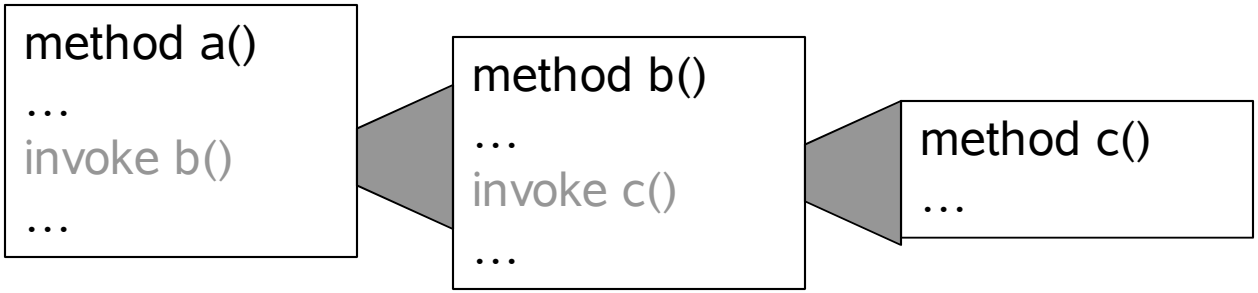
- There are ways to control inline expansion
- Inline sequences [HG03,BB04]
- Selectively inlining:
  - What if compile method a()?
  - What if compile method b()?



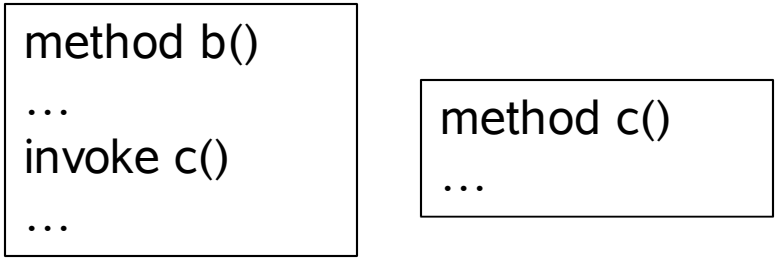
Code Expansion Control



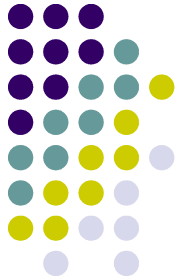
- Compile method a()
  - Inline methods b() and c()



- Compile method b()
  - No inlining



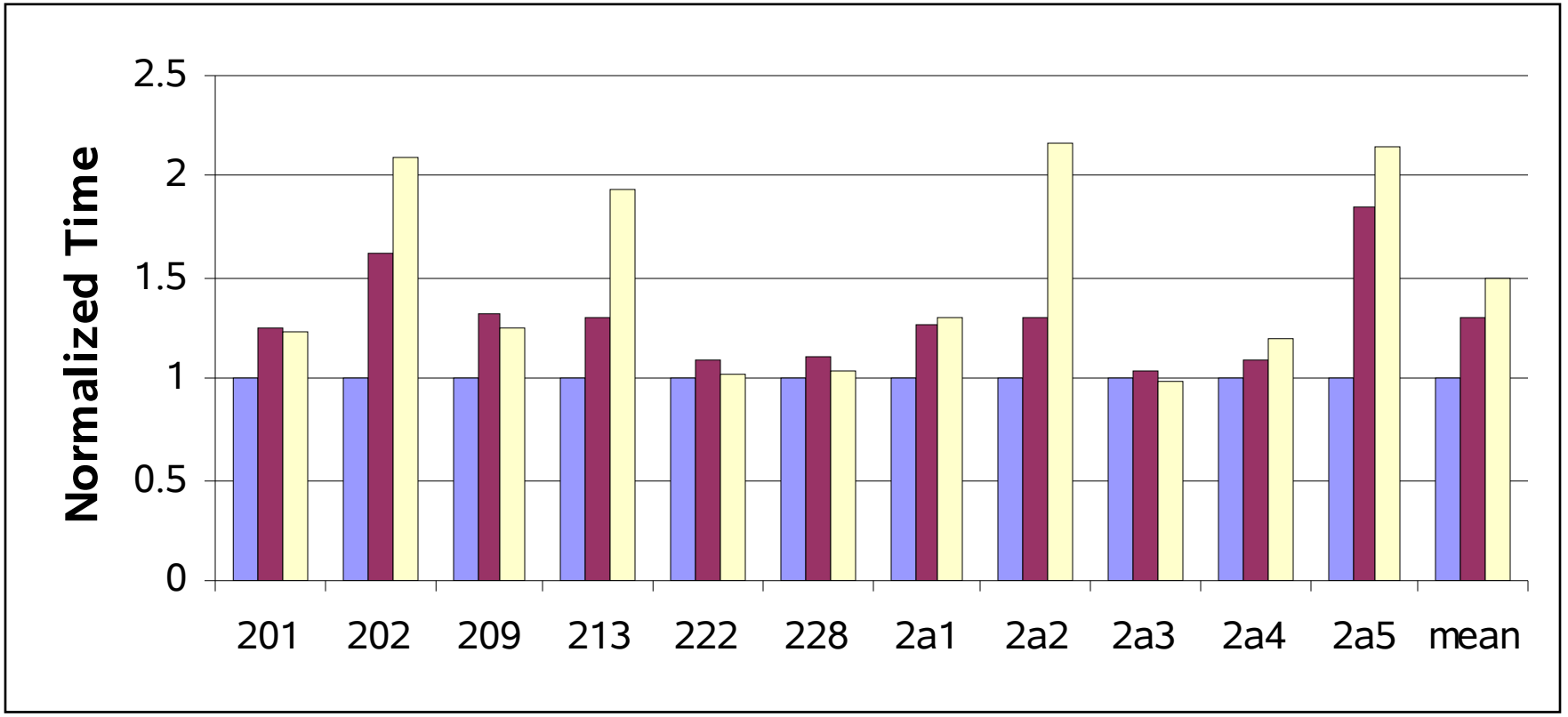
Results



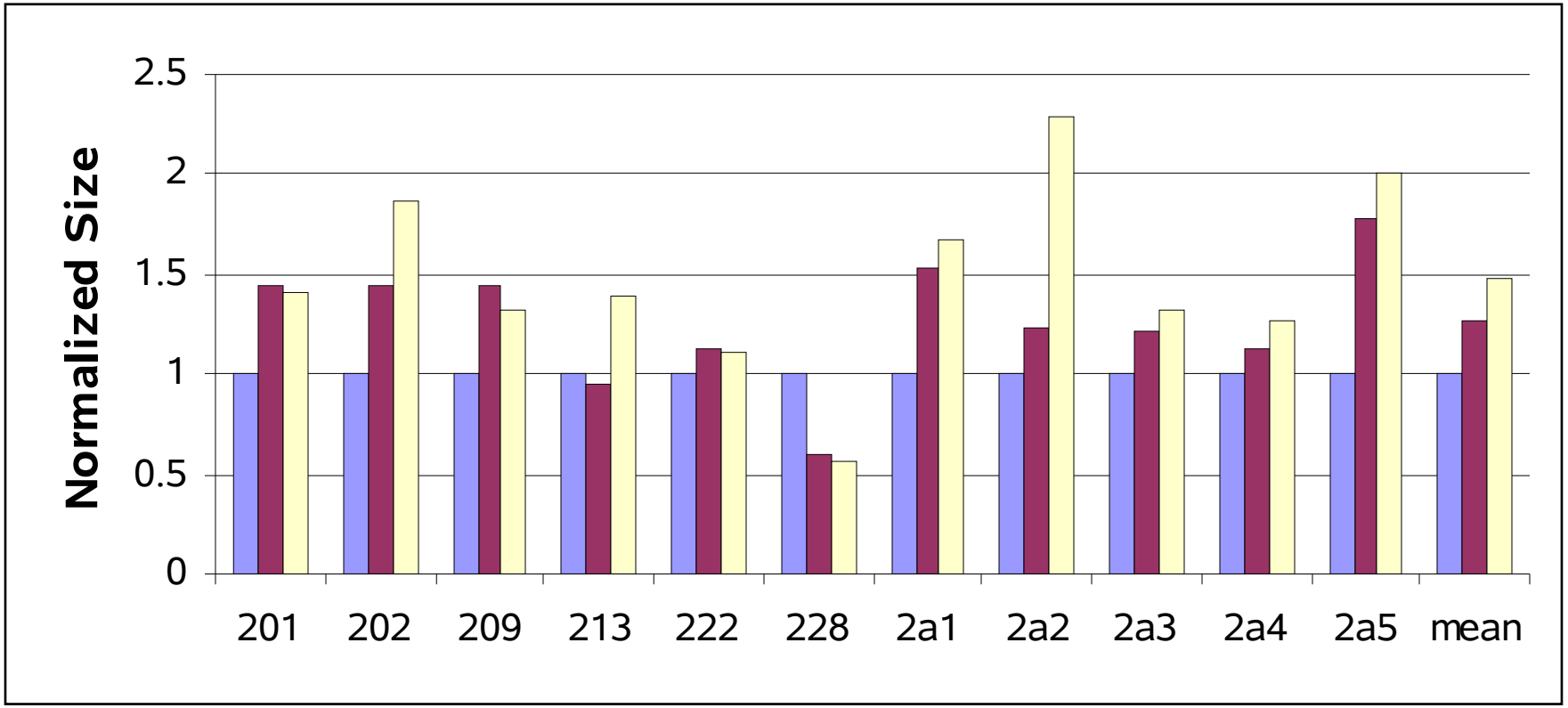
- Provide inline information to Jikes based on previous executions
- Compare our approach to two others:
  - Inline information provided by the Adaptive system of Jikes
  - A greedy algorithm based on work by Arnold et al. [Arn00]
- Evaluate two approaches: Just in Time and Ahead of Time
- Measure overhead of system

JIT Inlining – Execution Time

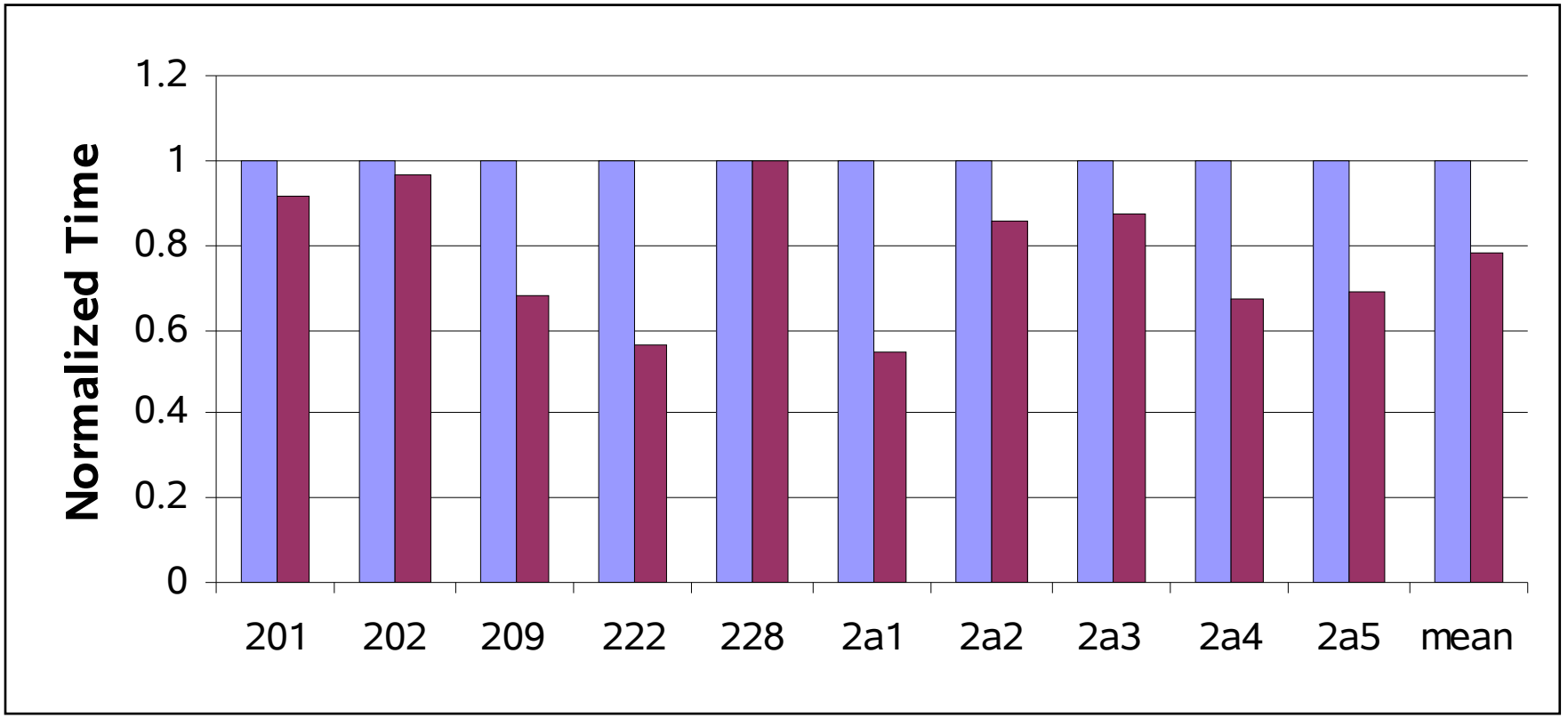
JIT Inlining – Compilation Time



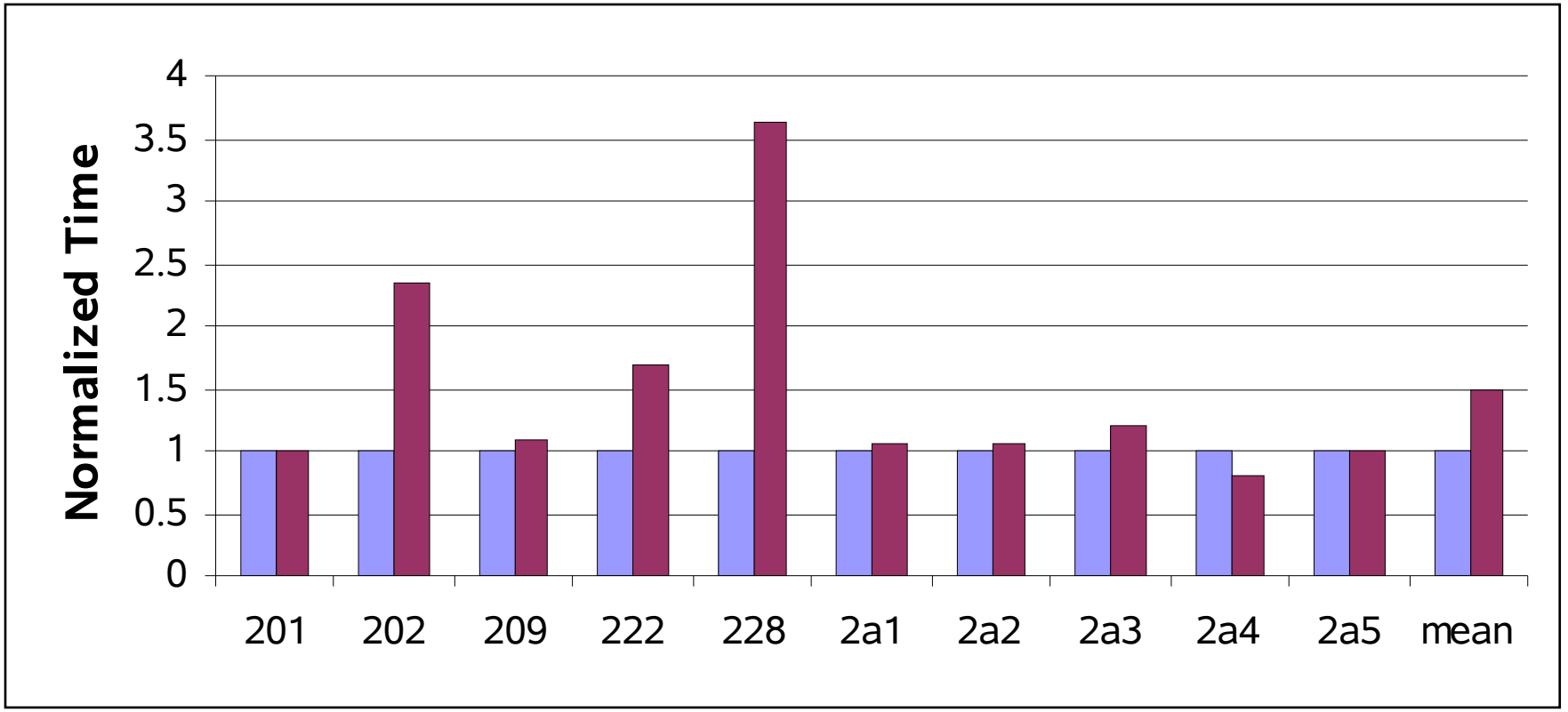
JIT Inlining – Code Expansion



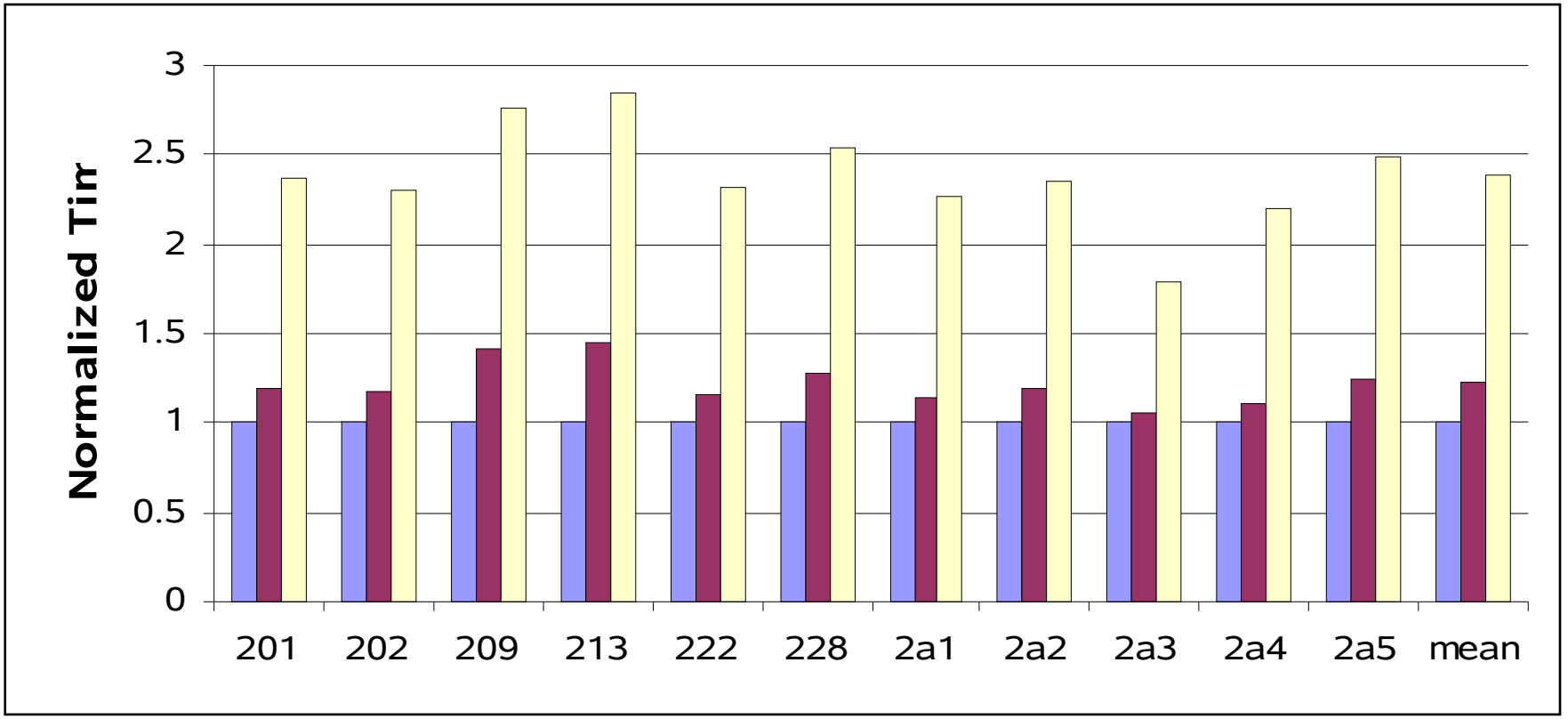
AOT Inlining – Execution Time



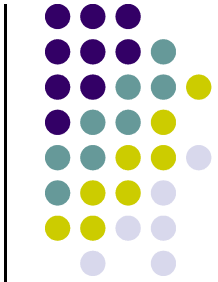
AOT Inlining – Compilation Time



Overhead

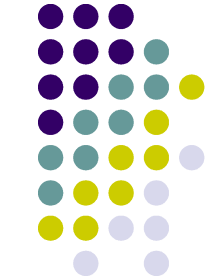

Normalized Tim
3
2.5
2
1.5
1
0.5
0
201
202
209
213
222
228
2a1
2a2
2a3
2a4
2a5
mean

Related Work



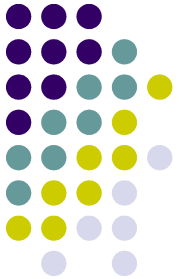
- Arnold et al. [Arn00]
  - Feedback-directed inlining in Java
  - Collected edge counts at method invocations
  - Used a greedy algorithm to select inlines that maximize invocations relative to code expansion
- Dynamo [BDB99]
  - Trace collection system
  - PA-RISC architecture
  - Assembly Instructions
  - Compiled traces

Conclusions



- Traces are beneficial for inlining:
  - Decreased execution time compared to one approach
  - Decrease competitive with another approach
  - Increases compilation time and code size
- A potential avenue of future research

Future Work



- Different trace collection strategies
- Trace based compilation and execution
- Reduction of code size
- Application of traces to other optimizations
- Usage of an online feedback directed system

References

- [MSD00] Matthew Arnold, Stephen Fink, David Grove, Michael Hind, and Peter F. Sweeney. Adaptive optimization in the Jalapeno JVM. ACM SIGPLAN Notices, 35(10):47-65, 2000.
- [Mic03] Michael Cierniak et al. The open runtime platform: A flexible high-performance managed runtime environment. Intel Technology Journal, February 2003.
- [HG03] Kim Hazelwood and David Grove. Adaptive online context-sensitive inlining. International Symposium on Code Generation and Optimization, p 253-264, 2003.
- [BB04] Bradel, B.J.: The use of traces in optimization. Master's thesis, University of Toronto (2004).
- [Arn00] Matthew Arnold et al: A comparative study of static and profile-based heuristics for inlining. SIGPLAN Workshop on Dynamic and Adaptive Compilation and Optimization. (2000) 52-64.
- [BDB99] Vasanth Bala, Evelyn Duesterwald, and Sanjeev Banerjia. Transparent dynamic optimization: The design and implementation of dynamo. HP Laboratories Technical Report HPL1999 –78.

AOT – Compilation Time (Wall Time)