To Inline or Not to Inline?

Enhanced Inlining Decisions

Peng Zhao and José Nelson Amaral

Department of Computing Science
University of Alberta

Oct 4, 2003

1 – Outline

- ♠ Revisit the inlining technique in ORC
- ♠ Adapt aggressiveness for different benchmarks
- ♠ Introduce cycle density heuristics to avoid unnecessary code bloat
- ♠ Investigate the unexplored potential of inlining and motivate further research

2 – Inlining

♠ Inlining replaces a call site with the function body of the callee

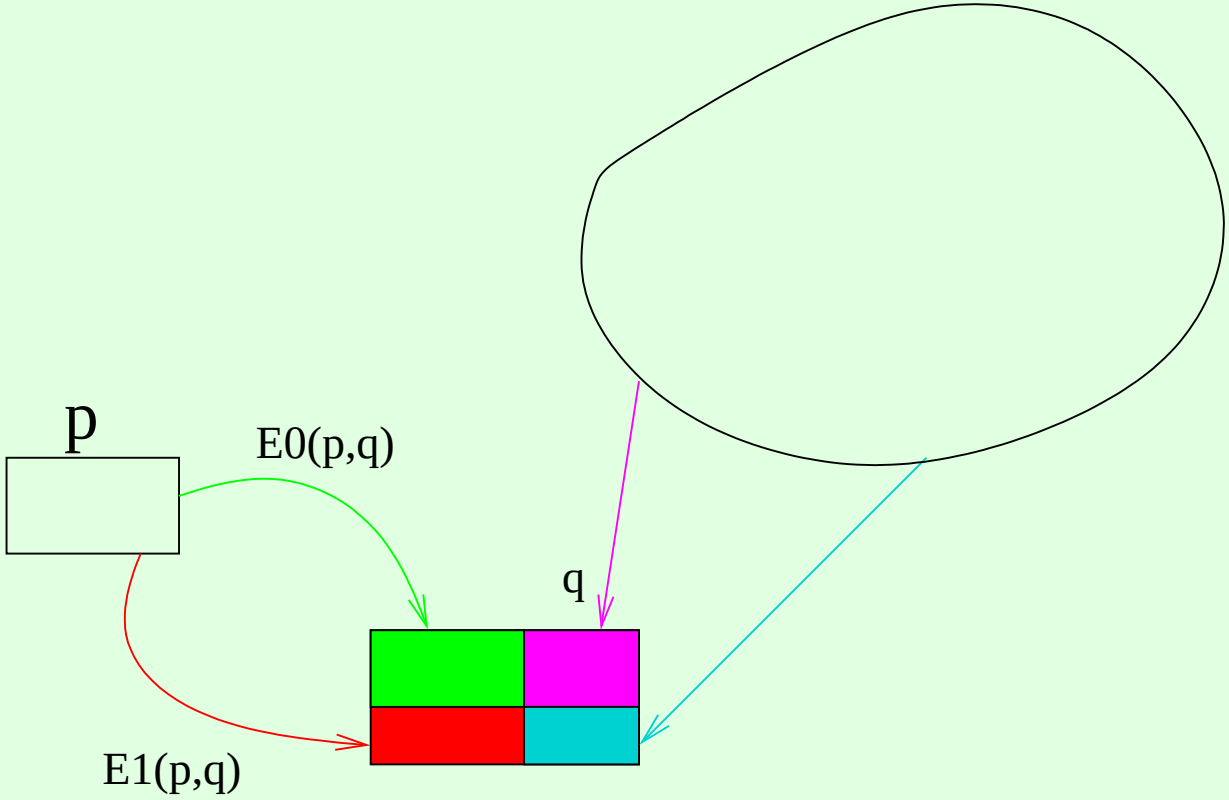
3 – Inlining Advantages

- ♠ Function invocation overhead eliminated
- ♠ Larger compiler analysis scope
- ♠ Better memory subsystem efficiency (cache, register)

4 – Inlining Disadvantages

- ♣ Code bloat
- ♣ More resources and time required for compilation
- ♣ Possible performance degradation

5 – Which call sites to inline?



6 – Current Inlining Heuristics in ORC

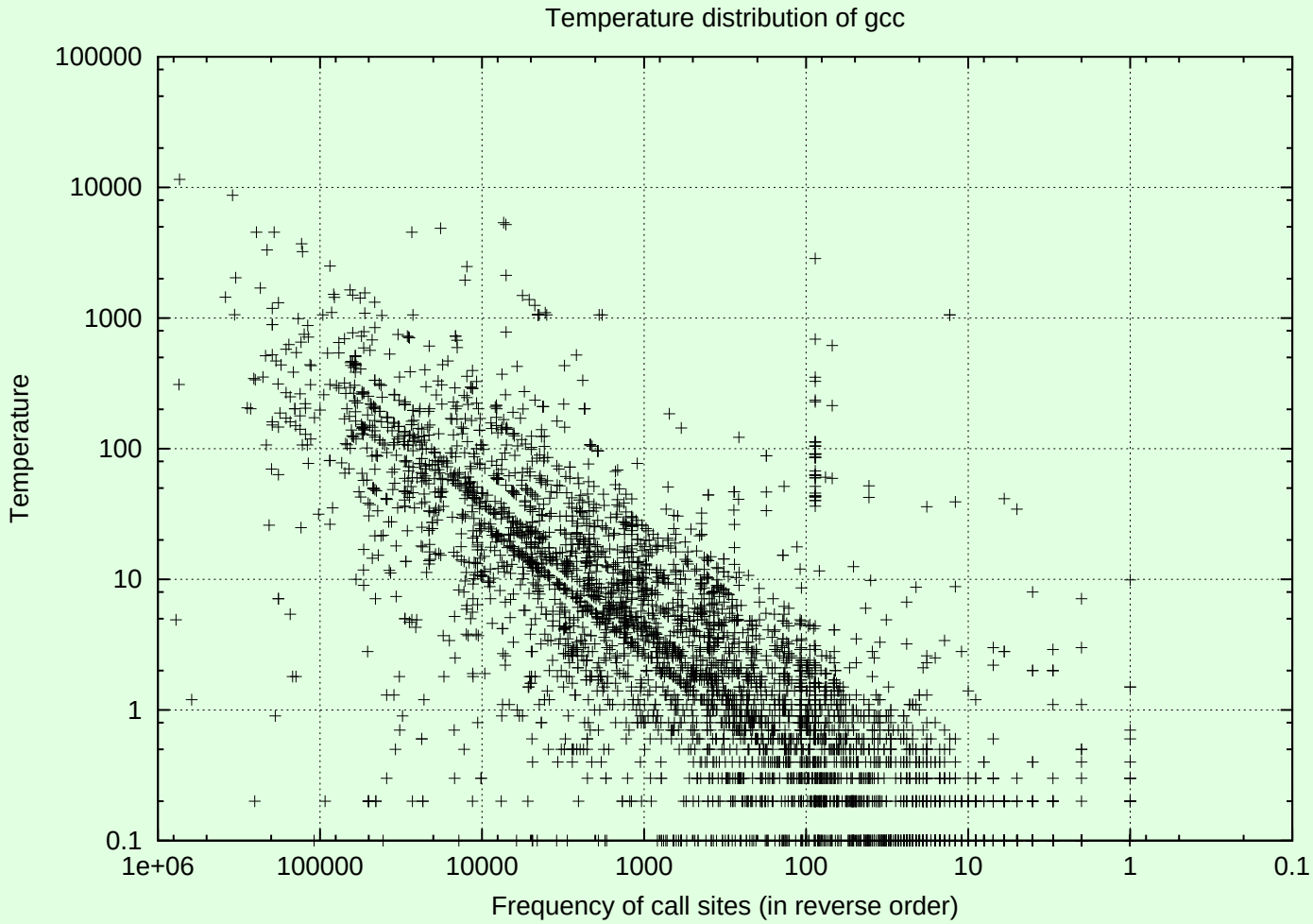
$$temperature_{E_i(p,q)} = \frac{cycle\_ratio_{E_i(p,q)}}{size\_ratio_q} \quad (1)$$

where:

$$cycle\_ratio_{E_i(p,q)} = \frac{cycle\_count_q}{Total\_cycle\_count} \times \frac{freq_{E_i(p,q)}}{freq_q} \quad (2)$$

$$size\_ratio_q = \frac{size_q}{Total\_application\_size} \quad (3)$$

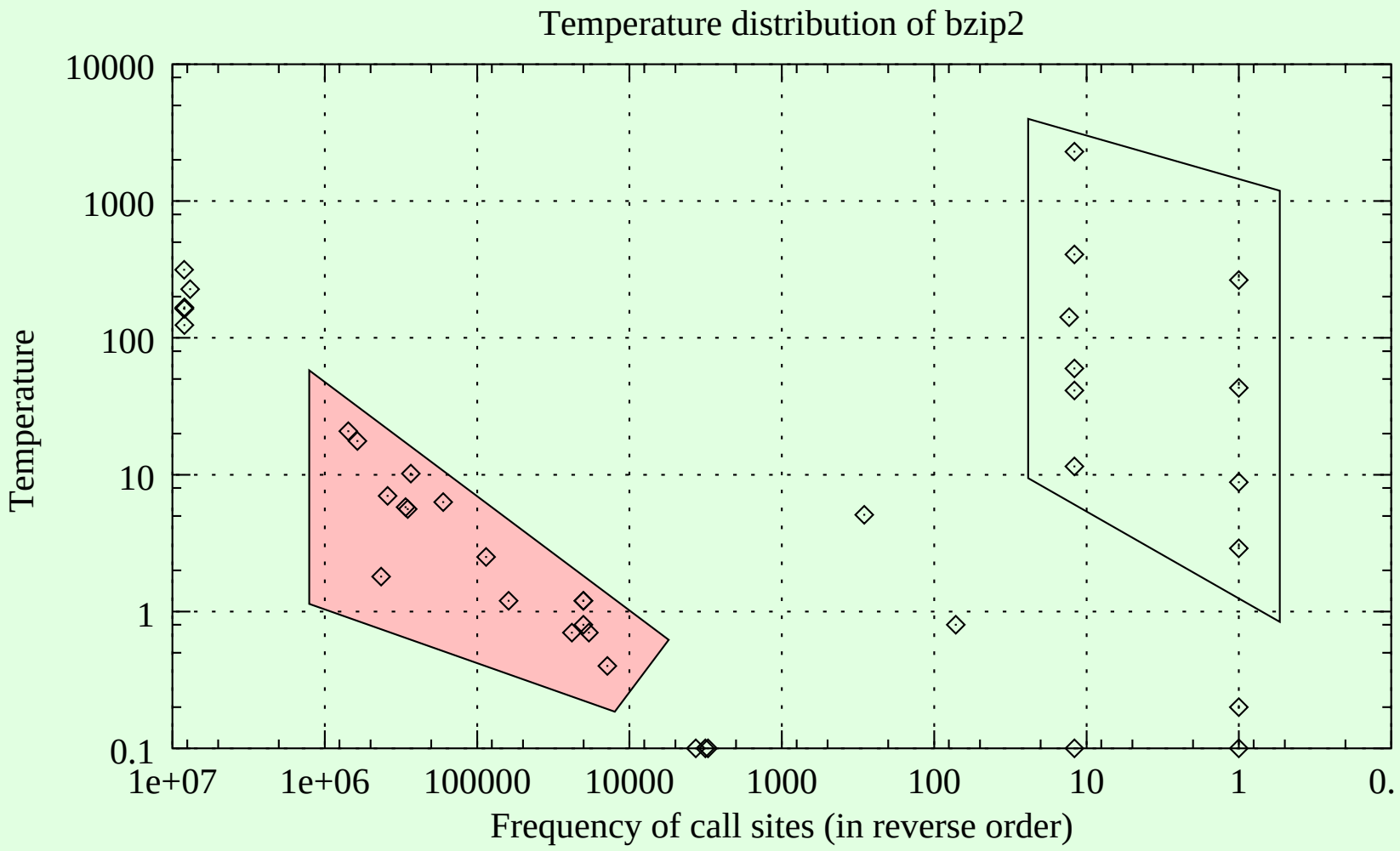
Figure 1: Example: Temperature Distribution of GCC



7 – Problems of Inlining Heuristics in ORC (1)

♠ Fixed temperature threshold is inflexible for different applications. Specially, it is too conservative for small applications.

Figure 2: Example: Temperature Distribution of BZIP2



8 – Adaptive inlining

♠ Adaptive inlining: we apply very aggressive inlining for small benchmarks and become conservative for large benchmarks.

♣ According to their estimated sizes, applicatinos are classified into 3 categories: large (above 250,000 AST nodes) , small (below AST 10,000 nodes) and medium (other) applications.

♣ We assign different temperature threshold for these three kinds of applications (120, 1, and 50, respectively).

9 – Problems of Inlining Heuristics in ORC (2)

♠ Fixed temperature threshold is inflexible for different applications. Especially, it is too conservative for small applications.

♠ Inlining of edges with high temperature but very low frequency (heavy edges), causing unnecessary code bloat.

10 – Why *Heavyfunctions* appears?

♠ *Heavyfunctions* are those cold functions with high-trip-count loops.

$$temperature_{E_i(p,q)} = \frac{cycle\_ratio_{E_i(p,q)}}{size\_ratio_q} \quad (4)$$

where:

$$cycle\_ratio_{E_i(p,q)} = \frac{freq_{E_i(p,q)}}{freq_q} \times \frac{cycle\_count_q}{Total\_cycle\_count} \quad (5)$$

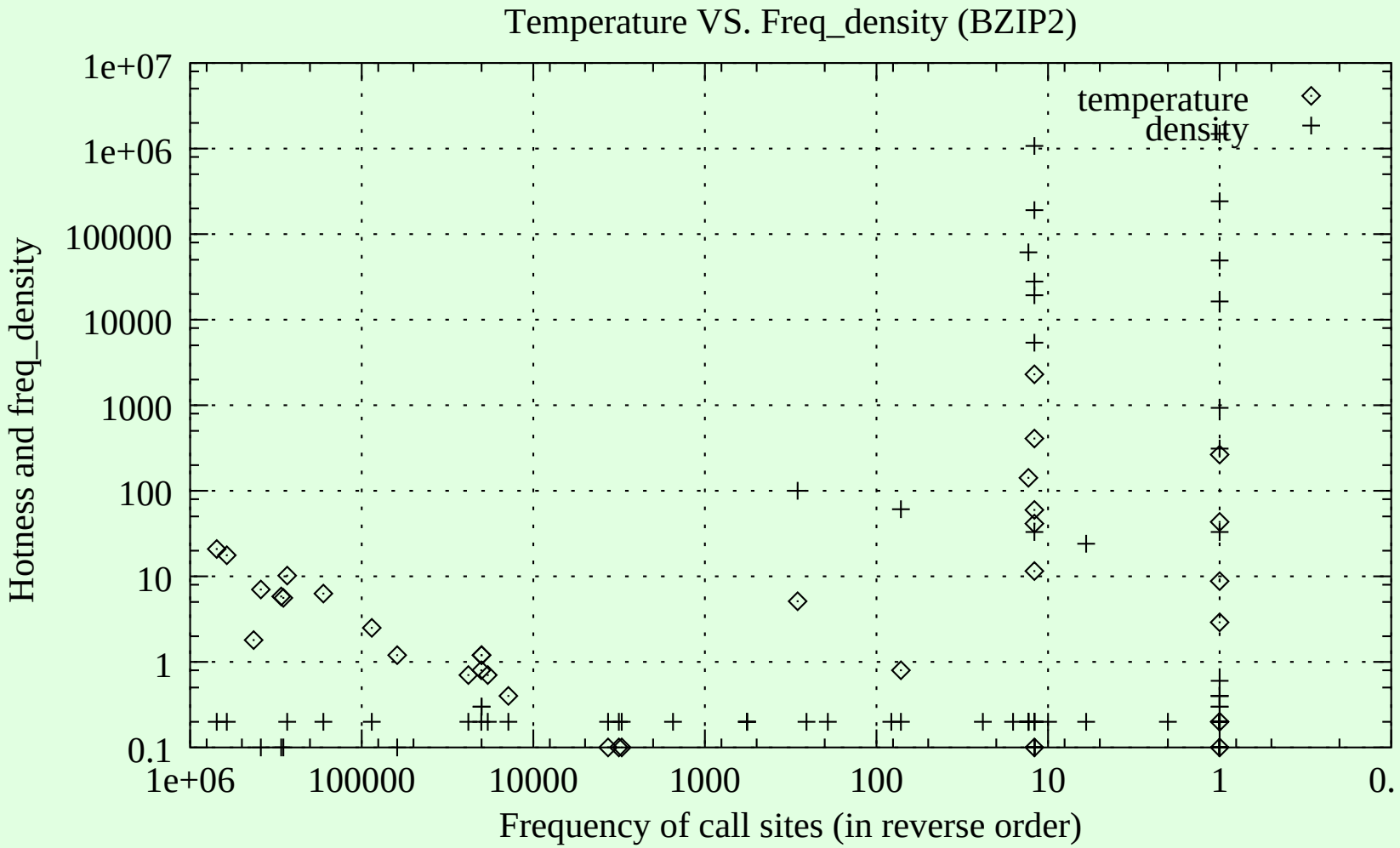
$$size\_ratio_q = \frac{size_q}{Tatal\_application\_size} \quad (6)$$

11 – Cycle_Density to Handle Heavy Functions

Cycle_Density means the average cycles spent in a function for each invocation. A very large $Cycle\_Density$ tells the function is a heavy function. Thus, we do not inline functions with high *Cycle_Density*.

$$cycle\_density_q = \frac{cycle\_count_q}{frequency_q} \qquad (7)$$

To Inline or Not to Inline? Enhanced Inlining Decisions



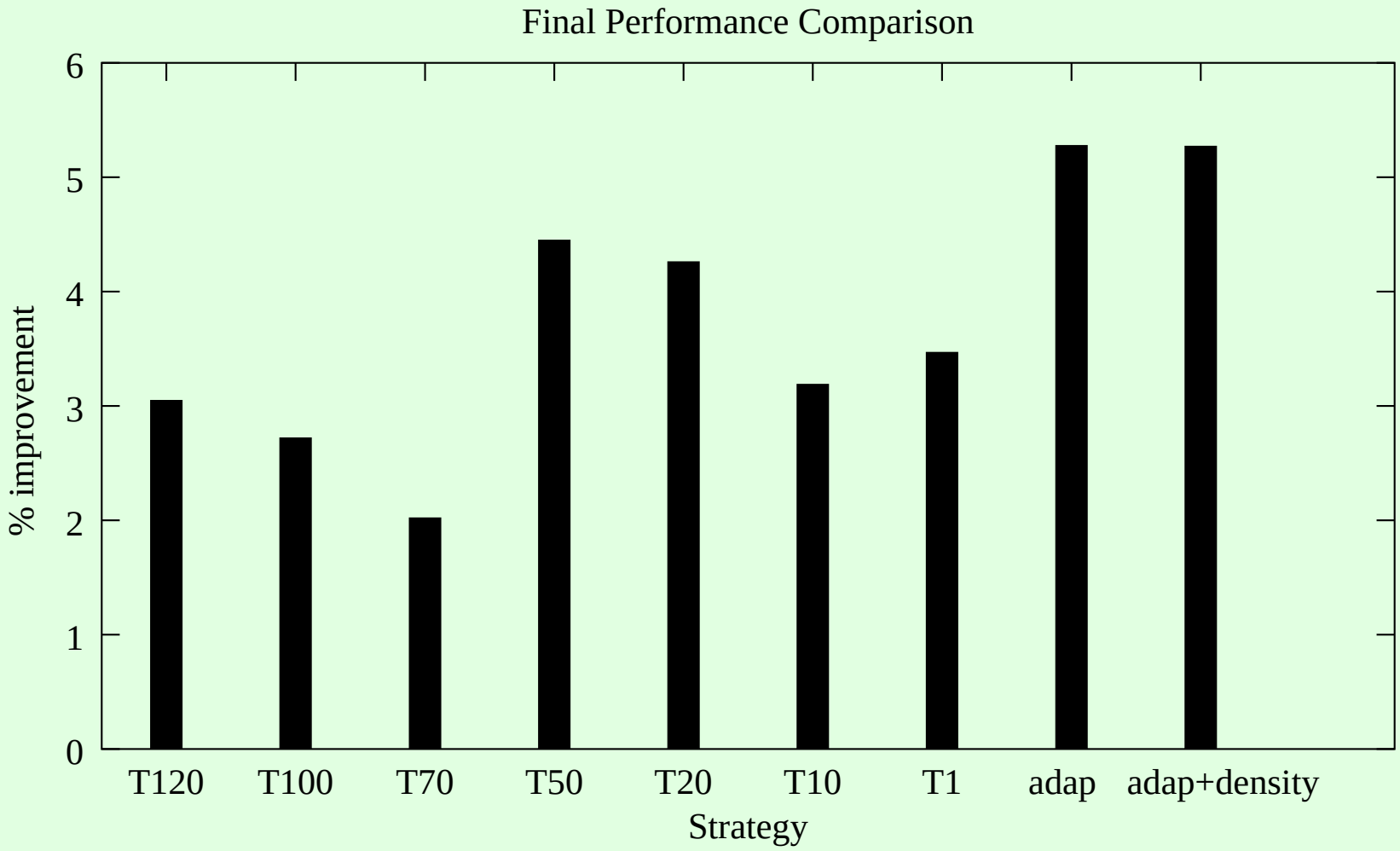
12 – Experimental Study

♠ Conducted on SPEC INT2000 benchmarks except EON

♠ Runtime: Itanium-I (733MHz, 1GB mem, RH-Linux 7.1 (2.4.21))

♠ Compilation: Dual-PIII (600MHz, 512 mem, RH-linux 7.2 (2.4.9))

13 – Performance Comparison



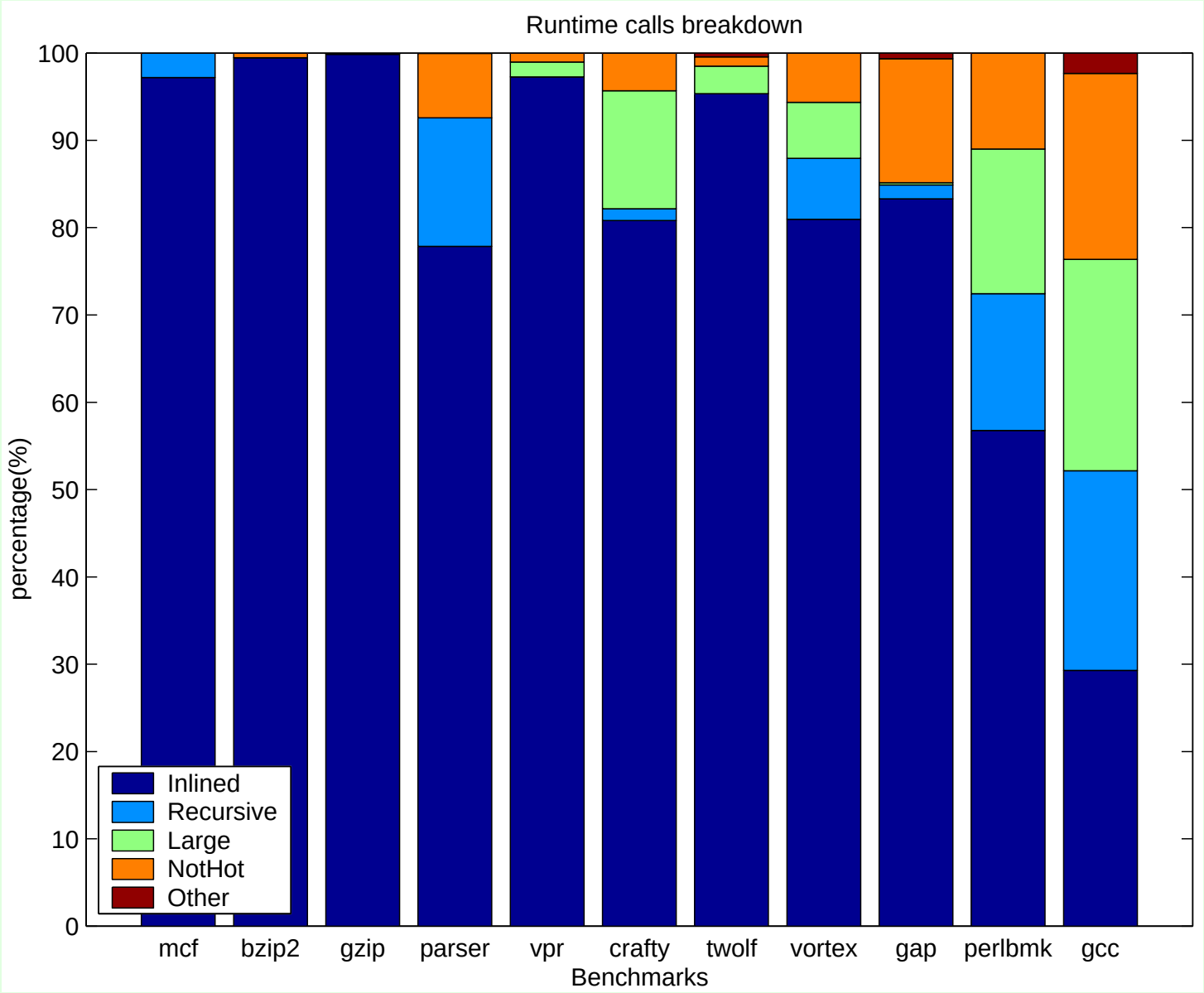
14 – Effect of Cycle_Density

	Executable Size		Compilation Time	
	adaptive	adap+density	adaptive	adap+density
average	+21.9%	+14.8%	+34.3%	+24.0%

* Comparison is based on the same configuration without inlining.

How far can we go?

To Inline or Not to Inline? Enhanced Inlining Decisions



15 – Future work

- ♠ Partial inlining on high level intermediate representation
- ♠ Unroll recursive function calls

Thank you very much!