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
♠ 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?

\[ \text{E0}(p, q) \quad \text{E1}(p, q) \]

\( p \quad q \)
6 – Current Inlining Heuristics in ORC

\[
\text{temperature}_{E_i(p,q)} = \frac{\text{cycle\_ratio}_{E_i(p,q)}}{\text{size\_ratio}_q} \quad (1)
\]

where:

\[
\text{cycle\_ratio}_{E_i(p,q)} = \frac{\text{cycle\_count}_q}{\text{Total\_cycle\_count}} \times \frac{\text{freq}_{E_i(p,q)}}{\text{freq}_q} \quad (2)
\]

\[
\text{size\_ratio}_q = \frac{\text{size}_q}{\text{Total\_application\_size}} \quad (3)
\]
Figure 1: Example: Temperature Distribution of GCC
Fixed temperature threshold is inflexible for different applications. Specially, it is too conservative for small applications.
Figure 2: Example: Temperature Distribution of BZIP2

Temperature distribution of bzip2

Temperature

Frequency of call sites (in reverse order)
Adaptive inlining: we apply very aggressive inlining for small benchmarks and become conservative for large benchmarks.

According to their estimated sizes, applications 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).
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 *Heavy functions* appear?

♦ *Heavy functions* are those cold functions with high-trip-count loops.

\[
\text{temperature}_{E_i(p,q)} = \frac{\text{cycle\_ratio}_{E_i(p,q)}}{\text{size\_ratio}_q}
\]  

(4)

where:

\[
\text{cycle\_ratio}_{E_i(p,q)} = \frac{\text{freq}_{E_i(p,q)}}{\text{freq}_q} \times \frac{\text{cycle\_count}_q}{\text{Total\_cycle\_count}}
\]  

(5)

\[
\text{size\_ratio}_q = \frac{\text{size}_q}{\text{Total\_application\_size}}
\]  

(6)
To Inline or Not to Inline? Enhanced Inlining Decisions

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*.

\[
cycle_{density}_q = \frac{cycle\_count_q}{frequency_q}
\]  

(7)
To Inline or Not to Inline? Enhanced Inlining Decisions

Temperature VS. Freq_density (BZIP2)

Hotness and freq_density

Frequency of call sites (in reverse order)
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))
Final Performance Comparison

% improvement

Strategy

T120 T100 T70 T50 T20 T10 T1 adap adap+density
### 14 – Effect of Cycle_Density

<table>
<thead>
<tr>
<th></th>
<th>Executable Size</th>
<th>Compilation Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>adaptive</td>
<td>adap+density</td>
</tr>
<tr>
<td>average</td>
<td>+21.9%</td>
<td>+14.8%</td>
</tr>
<tr>
<td></td>
<td>adaptive</td>
<td>adap+density</td>
</tr>
<tr>
<td></td>
<td>+34.3%</td>
<td>+24.0%</td>
</tr>
</tbody>
</table>

* Comparison is based on the same configuration without inlining.
How far can we go?
To Inline or Not to Inline? Enhanced Inlining Decisions

![Runtime calls breakdown chart]

- Inlined
- Recursive
- Large
- NotHot
- Other

Benchmarks: mcf, bzip2, gzip, parser, vpr, crafty, twolf, vortex, gap, perlbmk, gcc

Percentage (%)

Runtime: 0-100
15 – Future work

♠ Partial inlining on high level intermediate representation

♠ Unroll recursive function calls
Thank you very much!