Interprocedural IR Outlining For Code Size

River Riddle
Sony Interactive Entertainment - Tools & Technology
Why?

- Code Size is Important
  - Embedded Systems!
- Outlining is Low Hanging Fruit
Outlining For Code Size

Reduce code size by finding common code sequences and extracting them to separate functions.

• Easy Examples:
  – Function calls with constant parameters.
    • fprintf(stderr, “some msg”);
    • Error handling, memory allocation, intrinsics, API calls, etc.
  – Macros
  – Templates
  – Inline Functions
Outlining at the IR Level

- Value Number based on Semantic Equivalence
  - “Is this the same operation?” vs “Does this compute the same value?”
- Common Substring Query
- Candidate Analysis & Optimization
  - Parameterize the different values used.
- Cost Modeling
- Pruning & Outlining

```c
%17 = tail call i32 @strcmp(i8* %11, i8* %6)
%18 = icmp eq i32 %17, 0
....
%33 = tail call i32 @strcmp(i8* %22, i8* %4)
%34 = icmp eq i32 %33, 0

%17 = outlinedFn(i8* %11, i8* %6)
....
%33 = outlinedFn(i8* %22, i8* %4)

define i1 @outlinedFn(i8*, i8*) {
  %3 = tail call i32 @strcmp(i8* %0, i8* %1)
  %4 = icmp eq i32 %3, 0
  ret i1 %4
}
```
Choosing IR over Machine Level

Con - Inaccurate Cost Model
- Instructions folded during lowering
- Cost Visibility
  - May be additional cost that is only visible when lowering.
  - E.g. global addressing, int imm., parameter passing.

Pros
- More Opportunity
  - Less noise, Less restrictive equivalence.
- Pipeline Flexibility
- Target Independent
Performance

Improvement over Clang Oz

* Early - Pre Inlining.  * Late - Pre ISel

Spec 2006 Code Size % Geomean Improvement

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Early</th>
<th>Late</th>
<th>Early+ Late</th>
</tr>
</thead>
<tbody>
<tr>
<td>X86-64</td>
<td>3.24</td>
<td>3.02</td>
<td>4.45</td>
</tr>
<tr>
<td>X86</td>
<td>3.43</td>
<td>4.20</td>
<td>5.39</td>
</tr>
<tr>
<td>Arm</td>
<td>3.43</td>
<td>4.10</td>
<td></td>
</tr>
<tr>
<td>Arm-thumb</td>
<td>2.82</td>
<td>3.58</td>
<td></td>
</tr>
<tr>
<td>AArch64</td>
<td>3.62</td>
<td>3.42</td>
<td></td>
</tr>
</tbody>
</table>

Spec 2006 Code Size % Total Size Reduction

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Early</th>
<th>Late</th>
<th>Early+ Late</th>
</tr>
</thead>
<tbody>
<tr>
<td>X86-64</td>
<td>3.39</td>
<td>4.38</td>
<td></td>
</tr>
<tr>
<td>X86</td>
<td>3.28</td>
<td>5.29</td>
<td></td>
</tr>
<tr>
<td>Arm</td>
<td>3.35</td>
<td>4.55</td>
<td></td>
</tr>
<tr>
<td>Arm-thumb</td>
<td>3.29</td>
<td>4.13</td>
<td></td>
</tr>
<tr>
<td>AArch64</td>
<td>3.64</td>
<td>4.68</td>
<td></td>
</tr>
</tbody>
</table>
Outlining the Future

• Different Outlining Methods
  – Existing
    • Post RA Machine Level
      – Currently available in-tree for X64, AArch64
    • IR Level
  – Future?
    – Pre RA Machine Level
    – Region based

• Abstracting Shared Logic
  – Utilities: Candidate Selection, Pruning, etc.
  – Relaxed equivalency analysis is mostly IR agnostic.
More Info

RFC 1: http://lists.llvm.org/pipermail/llvm-dev/2017-July/115666.html
RFC 2: http://lists.llvm.org/pipermail/llvm-dev/2017-September/117153.html
Full Performance: https://goo.gl/ZBjHCG
Thank You!