SMART LOOP BREAKER CHOICE

Haskell Implementors Workshop 2011 - Tokyo
Bas den Heijer
- Master’s student at Utrecht University
- Mail S.K.denHeijer@students.uu.nl / 6.keer.9@gmail.com
- Skype debasfoon
Hello

Bas den Heijer

• Master’s student at Utrecht University
• Mail  S.K.denHeijer@students.uu.nl / 6.keer.9@gmail.com
• Skype  debasfoon
Thesis supervised by Atze Dijkstra + Hans Bodlaender

Software Technology
- Functional Programming
- Compilers (UHC)
- Optimisation

Algorithms
- Big Oh’s
- Math
- NP-Completeness
- Fixed Parameter Tractability
- Treewidth
LOOP BREAKING
LIKE A BOSS
Inlining looking like a graph

\[ f = g \]
\[ g = x + y \]
Inlining looking like a graph

\[ f = g \]
\[ g = x + y \]
\[ f = x + y \]
map f [] = []
map f (x:xs) = f x : map f xs
\[
\text{map } f \ [] = [] \\
\text{map } f \ (x:xs) = f \ x : \text{map } f \ xs
\]
map f [] = []
map f (x:xs) = f x : map f xs

map f [] = []
map f (x:xs) = f x : (case xs of [] -> [];
y:ys -> f y : map f ys) f xs

map f [] = []
map f (x:xs) = f x : (case xs of [] -> [];
y:ys -> f y : (case ys of [] -> [];
z:zs -> f z : map f zs) f ys) f xs
\[ \text{map } f \ [\] = [] \]
\[ \text{map } f \ (x:xs) = f\ x : \text{map } f\ xs \]
\[ \text{map } f \ [\] = [] \]
\[ \text{map } f \ (x:xs) = f\ x : (\text{case } xs \text{ of } [] \rightarrow []; y:ys \rightarrow f\ y : \text{map } f\ ys) \ f\ xs \]
\[ \text{map } f \ [\] = [] \]
\[ \text{map } f \ (x:xs) = f\ x : (\text{case } xs \text{ of } [] \rightarrow []; y:ys \rightarrow f\ y : (\text{case } ys \text{ of } [] \rightarrow []; z:zs \rightarrow f\ z : \text{map } f\ zs) \ f\ ys) \ f\ xs \]
Loop breakers

- Choose a breaker on every loop
- Don’t inline loop breakers
Loop breakers

• Choose a breaker on every loop
• Don’t inline loop breakers
Loop breakers

- Choose a breaker on every loop
- Don’t inline loop breakers
Loop breakers

Goals
• Don’t break variables that would be nice to inline
• Pick as few loop breakers as possible
GHC loop breaker heuristic

1. Decompose into strongly connected components
2. For each component with a cycle:
   A. Pick a node and make it a loop breaker
   B. If still cyclic, repeat from step 1

• Don’t pick a node with score $n$ if nodes with score $< n$ are still available
• For each score: after 2 random picks just make all nodes of that score loop breaker
So, the loop breaker heuristic...

Can we do better?

Can we do so quickly?

Do programs actually benefit?
In the mathematical discipline of graph theory, a feedback vertex set of a graph is a set of vertices whose removal leaves a graph without cycles.

- Wikipedia

- Applications in deadlock-mitigation, chip-design...
- NP-Complete
nofib/real/bspt
nofib/real/bspt
nofib/real/anna
nofib/real/*

30491 nodes
98218 edges
29066 strongly conn. components
98% of which are singletons
52% of the rest are dictionary-nests
85 nodes in largest scc
24 scc's larger than 10 nodes

GHC uses 2085 loop breakers, only 1754 are needed
So, the loop breaker heuristic...

Can we do better? Yes

Can we do so quickly?

Do programs actually benefit?
The exact algorithm

1. Split up in SCCs (strongly connected components)
2. Do the *priowiggle* to convert scores to blacked out nodes
3. Apply a few reduction rules
4. Branch & bound
The priowiggle

- Goal: black out nodes (make non-breaker) of high score, while making sure that it’s still possible to break all cycles

1. Find the lowest score $s$ such that breaking every node with score $\leq s$ results in an acyclic component
2. Black out all nodes with score $> s$
The priowiggle

- Goal: black out nodes (make non-breaker) of high score, while making sure that it’s still possible to break all cycles

1. Find the lowest score $s$ such that breaking every node with score $\leq s$ results in an acyclic component
2. Black out all nodes with score $> s$
Reduction rules

• Remove duplicate edges
• Keep splitting into SCCs
• Break self-loops

• Shortcut degree two

• Fix non-breaker cycles
- Remove duplicate edges
- Keep splitting into SCCs
- Break self-loops

- Shortcut degree two

- Fix non-breaker cycles
Compile time nofib/real

seconds

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>50</th>
<th>60</th>
<th>70</th>
</tr>
</thead>
<tbody>
<tr>
<td>GHC -O1</td>
<td>60</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exact FVS</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
So, the loop breaker heuristic...

Can we do better?  Yes

Can we do so quickly?  Yes

Do programs actually benefit?
nofib-analyse

- mode=slow
- -O1
- 25 runs

<table>
<thead>
<tr>
<th></th>
<th>Size</th>
<th>Allocs</th>
<th>Runtime</th>
<th>Elapsed</th>
<th>TotalMem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min</td>
<td>-0.0%</td>
<td>-4.9%</td>
<td>-6.4%</td>
<td>-6.5%</td>
<td>+0.0%</td>
</tr>
<tr>
<td>Max</td>
<td>+0.1%</td>
<td>+0.0%</td>
<td>+1.5%</td>
<td>+3.0%</td>
<td>+0.0%</td>
</tr>
<tr>
<td>Geometric mean</td>
<td>+0.0%</td>
<td>-0.1%</td>
<td>-0.4%</td>
<td>-0.0%</td>
<td>-0.0%</td>
</tr>
</tbody>
</table>
What’s going on here?

• Maybe loop breaker choice isn’t important
  • More opportunities, but inliner ignores them (make more aggressive?)

• Blame the benchmark
  • Tests are small: 48/91 programs take less than 200 ms, are ignored in totals
  • Maybe improved components are not covered much by tests

• Untested advantages
  • More flexible scoring possible: not just priorities but real scores (for example from a profile)
So, the loop breaker heuristic...

Can we do better?  Yes

Can we do so quickly?  Yes

Do programs actually benefit?  No.. not yet
Doing it on the edges
Doing it on the edges
Doing it on the edges

- More accurate portrayal of costs
- Call-site aware
  - Opportunity for more fine-grained scores

\[
5x = \text{Tree } y \ z
\]

\[
\text{case } x \ of
\]

\[
\text{Tree } \_ \_ \ \rightarrow a
\]

\[
\text{Leaf } \_ \ \rightarrow b
\]

- \textit{DFun} special case is no longer necessary
Doing it on the edges

- Blackout feedback \( \text{arc} \) set
- Same trick, slightly different reduction rules

Optimum FAS is 44% smaller than GHC’s heuristic on nofib/real
So, the loop breaker heuristic...

Can we do better?  Yes

Can we do so quickly?  Yes

Do programs actually benefit?  No.. not yet

nodes

edges

??
spectral/boyer
spectral/boyer
spectral/boyer
spectral/boyer
spectral/boyer