Faster Haskell

Neil Mitchell

www.cs.york.ac.uk/~ndm
The Goal

• Make Haskell “faster”
  – Reduce the runtime
  – But keep high-level declarative style

• Full automatic - no “special functions”
  – Different from foldr/build, steam/unstream

• Whole program optimisation
  – But fast (developed in Hugs!)
Word Counting

- In Haskell

```haskell
main =
    print . length . words =<< getContents
```

- Very high level
- A nice “specification” of the problem

*Note: getContents reimplemented in terms of getchar*
And in C

```c
int main() {
    int i = 0, c, last_space = 1;
    while ((c = getchar()) != EOF) {
        int this_space = isspace(c);
        if (last_space && !this_space) i++;
        last_space = this_space;
    }
    printf("%i\n", i);
    return 0;
}
```

About 3 times faster than Haskell
Why is Haskell slower?

• Intermediate lists! (and other things)
  – GHC goes through 4Gb of memory – $O(n)$
  – C requires ~13Kb – $O(1)$

• `length . words =<< getContents`
  – `getContents` produces a list
  – `words` consumes a list, produces a list of lists
  – `length` consumes the outer list
Removing the lists

• GHC already has foldr/build fusion
  – \(\text{map } f (\text{map } g \ x) = \text{map } (f \ . \ g) \ x\)

• But getContents is trapped under IO
  – Much harder to fuse automatically
  – Don’t want to rewrite everything as foldr
  – Easy to go wrong (take function in GHC 6.6)
Supero: My Optimiser

- Fully automatic
  - No annotations, special functions

- Evaluate the program at compile time
  - Start at main, and execute

- Stop when you reach a primitive
  - The primitive is in the optimised program
main r
(print . length . words =<<< getContents) r
(getContents >>= print . length . Words) r
case getContents r of (# s, r #) -> ...
getChar >>= if c == 0 then return [] else ...
case getChar r of ...

• Have reached a case on a primitive
The new program

main r = case getChar r of
    (# c, r #) -> main2 c r

• Create main2, for the alternative
• Continue optimisation on the branches of the case, main2
• The evaluation mainly does inlining
  – Also case/case, case/ctor, let movement
Tying in the knot

• Each name in the new program corresponds to an expression in the old
  – main = print . length . words =<< getContents
  – main2 = the case alternative

• If you reach the same expression, use the same name – makes recursive call
sum x = case x of
    [] -> 0
    (x:xs) -> x + sum xs

range i n = case i > n of
    True -> []
    False -> i : range (i+1) n

main n = sum (range 0 n)
main n
sum (range 0 n)
main n = main2 0 n
    where main2 i n = sum (range i n)
case range i n of {[] -> 0; x:xs -> x + sum xs}
case (case i > n of {True -> []; False -> …}) of …
case i > n of {True -> 0
    ;False -> i + sum (range (i+1) n)}
tie back:
    main2 (i+1) n
main n = main2 i n

main2 i n = if i > n then 0 else i + main2 (i+1) n

- Lists have gone entirely
- Everything is now strict
- Using sum as foldl or foldl’ would have given accumulator version
Ensuring Termination

To make the optimisation terminate
  – Need to “hide” some information
  – Anything which is an accumulator
  – i.e. foldl’s 2\textsuperscript{nd} argument

Lots of possible termination criteria
  – Want to give good optimisation
  – But not blow up the size of the code
Termination Problems

• One theme – bound recursion depth
• Problem 1:
  – Some optimisations require ~5 recursive inlinings
  – 5 recursive inlinings blows up code a lot
• Problem 2:
  – Repeated application can square any bound
  – Bound of 5 can become a bound of 25!
Back to word counting

• What if we use Supero on the Haskell?
  – Compile using yhc, to Yhc.Core
  – Optimise, using Supero
  – Write out Haskell, compile with GHC

• GHC provides:
  – Strictness/unboxing
  – Native code generator
Problem 1: isSpace

• On GHC, isSpace is too slow (bug 1473)
  – C's isspace: 0.375
  – C's iswspace: 0.400
  – Char.isSpace: 0.672

• For this test, I use the FFI

SOLVED!
Problem 2: words

words :: String -> [String]
words s = case dropWhile isSpace s of
""  -> []
    s'  -> w : words s'
      where (w, s") = break isSpace s'

• Does two extra isSpace tests per word
• Better version in Yhc
Other Problems

- Wrong strictness information (bug 1592)
  - IO functions do not always play nice
- Badly positioned heap checks (bug 1498)
  - Tight recursive loop, where all time is spent
  - Allocates only on base case (once)
  - Checks for heap space every time
- Unnecessary stack checks
- Probably ~15% slowdown
Performance

• Now Supero+GHC is 10% faster than C!
  – Somewhat unexpected…
  – Can anyone guess why?

```c
while ((c = getchar()) != EOF)
    int this_space = isspace(c);
    if (last_space && !this_space) i++;
    last_space = this_space;
```
• Haskell encodes space/not in the program counter!
• Hard to express in C
The “wc” benchmark

- Charcount
- Linecount
- Wordcount

C
Supero
GHC
Haskell Benchmarks

• Working towards the nofib/nobench suite
  – Termination vs optimisation problem
  – Massively more complex
  – Much larger volumes of code

• Particular issues
  – The read function
  – Invoking a Haskell lexer to read an Int!
  – List comprehensions (as desugared by Yhc)
Conclusions

• Still lots of work to do before concluding!
  – Nobench is a priority

• Haskell can be both beautiful *and* fast

**Thanks to:** Simon Peyton Jones, Simon Marlow, Tim Chevalier for low-level GHC help