CSE 101 HW #6

Chris Aikman
Hugo Rivera

April 14, 2015

1 Programming Paradigm Assignment

1.1 Imperative

Example Language:
XXX
Application:
XXX
Application Appropriateness:
XXX

1.2 Functional

Example Language:
Haskell
Application:
Haskell has been used to program several web servers, most notably Snap and Yesod.
Snap: `http://snapframework.com/`
Yesod: `http://www.yesodweb.com/`
Application Appropriateness:
Haskell was an appropriate choice for designing these applications because it offers a powerful type system that protects the programmer from making trivial mistakes while still being flexible enough for real world use.

1.3 Object Oriented

Example Language:
C++
Application:
C++ is well known for being the language chosen to implement the two most common web browsers: Google Chrome and Mozilla Firefox.
Google Chrome: `http://www.google.com/chrome/`
Firefox: `https://www.mozilla.org/en-US/firefox/new/`

Application Appropriateness:
C++ was an appropriate choice for these web browsers because it gives the programmer a lot of power while offering the advantages of object oriented programming, which allows code to be smaller, more

modular and easier to manage. C++ has also been used, tested and evolved for decades making it extremely reliable when used correctly.

1.4 Logic

Example Language:
XXX
Application:
XXX
Application Appropriateness:
XXX

2 Parallel Matrix Multiply Assignment

2.1 Pseudocode

Haskell is a declarative language, thus this is declarative pseudocode.

```
a matrix is a 2D array of elements that can be multiplied and added

to multiply two matrices a (dimensions n by m), b (dimensions m by p):
    let the result be matrix c of dimensions
    let m = shared dimension of b and a
            if there is no shared dimension, throw an error

    map (ra ->
        map (cb ->
            sum [ra[i]*b[i] | i<-[0..m]])
        b.cols)
    a.rows
```

2.2 Haskell Code

```
1 module MatrixMult (multMatrix) where
2 import Data.List (tranpose)
3
4 multMatrix a b =
5     map (\ra ->
6         map (\cb ->
7             sum $ zipWith (*) ra cb) (transpose b)) a
```

2.3 Assembly Code Computation

Done using "ghc -S matrix_mult.hs" See the appendix.

2.4 Assembly Code Multiplication

This happens in line 52 after preparing the arguments. Haskell performs a jump the number library's integer multiplication method "jmp base_GHCziNum_zt_info" (instructions for Base.GHC.zahltimes or integer mult)

2.5 Number of Assembly Code Lines for Multiplication

It takes 6 lines of code starting at line 47 to gather the arguments and prepare the program for jumping to "base_GHCziNum_zt_info"

3 Appendix

```
1  .data
2          .align 8
3  .align 1
4  .globl __stginit_MatrixMult
5  .type __stginit_MatrixMult, @object
6  __stginit_MatrixMult:
7  .data
8          .align 8
9  .align 1
10 .globl MatrixMult_multMatrix_closure
11 .type MatrixMult_multMatrix_closure, @object
12 MatrixMult_multMatrix_closure:
13         .quad   MatrixMult_multMatrix_info
14         .quad   0
15 .text
16         .align 8
17         .long   SSg_srt-(sRo_info)+0
18         .long   0
19         .quad   1
20         .quad   4294967313
21 sRo_info:
22 .LcRD:
23         leaq -16(%rbp),%rax
```

```
24          cmpq %r15,%rax
25          jb .LcRE
26  .LcRF:
27          movq $stg_upd_frame_info,-16(%rbp)
28          movq %rbx,-8(%rbp)
29          movq 16(%rbx),%rax
30          movq %rax,%r14
31          movl $base_DataziList_transpose_closure,%ebx
32          addq $-16,%rbp
33          jmp stg_ap_p_fast
34  .LcRE:
35          jmp *-16(%r13)
36          .size sRo_info, .-sRo_info
37  .text
38          .align 8
39          .quad   1
40          .quad   17
41  sRl_info:
42  .LcRT:
43          leaq -16(%rbp),%rax
44          cmpq %r15,%rax
45          jb .LcRU
46  .LcRV:
47          movq $stg_upd_frame_info,-16(%rbp)
48          movq %rbx,-8(%rbp)
49          movq 16(%rbx),%rax
50          movq %rax,%r14
51          addq $-16,%rbp
52          jmp base_GHCziNum_zt_info
53  .LcRU:
54          jmp *-16(%r13)
55          .size sRl_info, .-sRl_info
56  .text
57          .align 8
58          .long   SSg_srt-(sRm_info)+8
59          .long   0
60          .quad   3
61          .quad   4294967312
62  sRm_info:
63  .LcRW:
64          leaq -16(%rbp),%rax
```

```
65          cmpq %r15,%rax
66          jb .LcRX
67  .LcRY:
68          addq $24,%r12
69          cmpq 856(%r13),%r12
70          ja .LcS0
71  .LcRZ:
72          movq $stg_upd_frame_info,-16(%rbp)
73          movq %rbx,-8(%rbp)
74          movq 16(%rbx),%rax
75          movq 24(%rbx),%rcx
76          movq 32(%rbx),%rbx
77          movq $sRl_info,-16(%r12)
78          movq %rax,(%r12)
79          leaq -16(%r12),%rax
80          movq %rbx,%rdi
81          movq %rcx,%rsi
82          movq %rax,%r14
83          movl $base_GHCziList_zzipWith_closure,%ebx
84          addq $-16,%rbp
85          jmp stg_ap_ppp_fast
86  .LcS0:
87          movq $24,904(%r13)
88  .LcRX:
89          jmp *-16(%r13)
90          .size sRm_info, .-sRm_info
91  .text
92          .align 8
93          .long    SSg_srt-(sRn_info)+8
94          .long    0
95          .quad    4294967301
96          .quad    2
97          .quad    12884901900
98  sRn_info:
99  .LcS1:
100 .LcS3:
101         addq $40,%r12
102         cmpq 856(%r13),%r12
103         ja .LcS5
104 .LcS4:
105         movq 7(%rbx),%rax
```

```
106             movq 15(%rbx),%rbx
107             movq $sRm_info,-32(%r12)
108             movq %rax,-16(%r12)
109             movq %rbx,-8(%r12)
110             movq %r14,(%r12)
111             leaq -32(%r12),%rbx
112             movq %rbx,%rsi
113             movq %rax,%r14
114             movl $base_DataziList_sum_closure,%ebx
115             jmp stg_ap_pp_fast
116     .LcS5:
117             movq $40,904(%r13)
118     .LcS2:
119             jmp *-8(%r13)
120             .size sRn_info, .-sRn_info
121     .text
122             .align 8
123             .long   SSg_srt-(sRp_info)+0
124             .long   0
125             .quad   4294967301
126             .quad   2
127             .quad   64424509452
128     sRp_info:
129     .LcS6:
130     .LcS8:
131             addq $48,%r12
132             cmpq 856(%r13),%r12
133             ja .LcSa
134     .LcS9:
135             movq 7(%rbx),%rax
136             movq 15(%rbx),%rbx
137             movq $sRo_info,-40(%r12)
138             movq %rbx,-24(%r12)
139             leaq -40(%r12),%rbx
140             movq $sRn_info,-16(%r12)
141             movq %rax,-8(%r12)
142             movq %r14,(%r12)
143             leaq -15(%r12),%rax
144             movq %rbx,%rsi
145             movq %rax,%r14
146             movl $base_GHCziBase_map_closure,%ebx
```

```
147         jmp stg_ap_pp_fast
148 .LcSa:
149         movq $48,904(%r13)
150 .LcS7:
151         jmp *-8(%r13)
152         .size sRp_info, .-sRp_info
153 .text
154         .align 8
155         .long   SSg_srt-(MatrixMult_multMatrix_info)+0
156         .long   0
157         .quad   12884901911
158         .quad   0
159         .quad   133143986191
160 .globl MatrixMult_multMatrix_info
161 .type MatrixMult_multMatrix_info, @object
162 MatrixMult_multMatrix_info:
163 .LcSb:
164 .LcSd:
165         addq $24,%r12
166         cmpq 856(%r13),%r12
167         ja .LcSf
168 .LcSe:
169         movq $sRp_info,-16(%r12)
170         movq %r14,-8(%r12)
171         movq %rdi,(%r12)
172         leaq -15(%r12),%rax
173         movq %rax,%r14
174         movl $base_GHCziBase_map_closure,%ebx
175         jmp stg_ap_pp_fast
176 .LcSf:
177         movq $24,904(%r13)
178 .LcSc:
179         movl $MatrixMult_multMatrix_closure,%ebx
180         jmp *-8(%r13)
181         .size MatrixMult_multMatrix_info, .-MatrixMult_multMatrix_info
182 .section .data
183         .align 8
184 .align 1
185 SSg_srt:
186         .quad   base_DataziList_transpose_closure
187         .quad   base_GHCziList_zzipWith_closure
```

```
188             .quad    base_DataziList_sum_closure
189             .quad    base_GHCziBase_map_closure
190             .quad    MatrixMult_multMatrix_closure
191     .section .note.GNU-stack,"",@progbits
192     .ident "GHC 7.8.3"
```