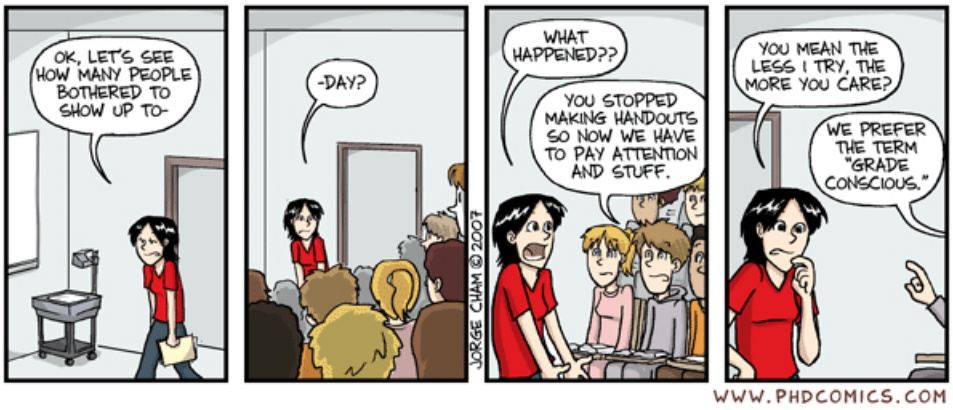
CS614: Advanced Compilers

End-Sem Exam (2.5 hours; 25 marks)

November 23^{rd}, 2023

Instructions:

- Write neat, clear and crisp answers. Start answering each **Q** on a new page.
- In case you make an assumption, write it down before the corresponding answer.
- Our exam is a relaxed 5-course Compilers Meal, with each course carrying 5 marks, along with a 1-mark bonus. Chew it all well!



Q1: DEPDEP SOUP [2+3]

(A) Match the columns such that the LHS information is useful while invoking the RHS pass:

(a) Liveness information	(i) Layout ordering
(b) Field types	(ii) Loop tiling
(c) Method-call counts	(iii) Garbage collection
(d) Cache-block size	(iv) JIT compilation

(B) Consider the code fragment shown below:

```
1.  LD R1, a
2.  LD R2, b
3.  SUB R3, R1, R2
4.  ADD R2, R1, R2
5.  ST a, R3
6.  ST b, R2
```

Firstly, draw the data dependence graph highlighting different kinds of dependences separately. Now assume a machine with one ALU resource (for the `ADD` and `SUB` operations) and one MEM resource (for the `LD` and `ST` operations). Assume that all operations require one clock except for `LD`, which requires two. Further, a `ST` on the same memory location can commence one clock after a `LD` on that location commences. Find a shortest schedule for the above piece of code on this machine.

Q2: REGISTER TIKKA [1+4]

(A) What could be the effect(s) of converting a program to SSA form before performing register allocation?

(B) Consider the following program snippet with eight variables `w1..w4` and `L1..L4`. Assuming each register is of 32 bits, perform bitwidth-aware register allocation for this program. First draw an extended interference graph, and then step by step pack variables to obtain a minimal-register configuration. Finally, rewrite the program to reflect bitwise register allocation.

```
main() {
   ...
   w1 = C1 & 0xffff;
   w2 = C2 & 0xffff;
   w3 = C3 & 0xffff;
   w4 = C4 & 0xffff;
   L1 = ((w1 * K1) >> 16);
   L2 = ((w2 * K3) >> 16);
   L3 = ((w3 * K3) >> 16);
   L4 = ((w4 * K4) >> 16);

   print L1 + L2 + L3 + L4;
}
```

Q3: LOOPY NOODLES [3+2]

(A) Optimize the following loops. Name the optimization (give a new name if we haven't learnt the same), and show the code transformation. More the (potential) improvement more will be the marks that you get!

```
for (i = 0; i < n; ++i) {
   if (x > y) {
      a[i] = b[i] * x;
   } else {
      a[i] = b[i] * y;
   }
}
```

(a)

```
    s = 0;
L1: if s > 0 goto L2;
    i += b;
    j = i * 4;
    x = M[j];
    s -= x;
    goto L1;
L2: i++;
    s += j;
    if i < n goto L1;
```

(b)

```
for (i = 0; i < n; ++i) {
   if (i%2 == 0) {
      Y[i] = Z[i];
   }
}
for (j = 0; j < n; ++j) {
   X[i] = Y[i];
}
```

(c)

(B) Molina has recently moved from heavy vector-processor programming to writing code for scalar (non-vector) machines. She found that for most loops the translation of vectorized instructions to the new machines is simple (a process called *scalarization*):

"Replace each vector instruction with a simple `for` loop that runs through the same indices as the range corresponding to the vector instruction."

For example, the code in (a) below gets converted to the one in (b):

```
A[1:200] = 2.0 * A[1:200];
```

(a)

```
for (i = 1; i <= 200; ++i) {
    A[i] = 2.0 * A[i];
}
```

(b)

However, Manoj claims that the scheme does not work for the instruction `A[2:201] = 2.0 * A[1:200]`.

Can you identify the problem with the simple scalarization algorithm above, and propose a new one that solves the problem? Also show the correctly transformed code for the second vector instruction.

Q4: NULLNESS BIRYANI [1+1.5+2.5]

As per Java's language specification, the run-time must perform a null-dereference check before performing an object dereference, and correspondingly throw a `NullPointerException` (NPE) if the reference does not point to a concrete object. For the following Java program snippet, answer the questions that follow:

```
1  class B {
2     X f;
3     B() {
4        f = new X();
5        f.g = new Y();
6        f.g.h = new Z();
7     }
8     void bar() {
9        B r1 = new B();
10       AList r2 = new AList();
11       r2.add(r1);
12       X x = r1.f;
13       Y y = x.g;
14       Z z = y.h;
15    }
16 }
```

```
17 class X { Y g; }
18 class Y { Z h; }
19 class Z { ... }
20
21 class AList {
22    Object[] arr;
23    ...
24    /*Assume the code compiles,
25      and ignore array bounds.*/
26    Alist() {
27       arr = new Object[100];
28    }
29    void add(Object e) {
30       arr[size++] = elem;
31    }
32 }
```

(A) List all the statement numbers that perform an object dereference.

(B) While compiling a Java class, assume that the compiler does not have access to the code of other classes. List the subset of dereferences from part (A) that would definitely *not* throw an NPE.

(C) Recall that JIT compilers typically choose efficiency of compilation over precision of the underlying passes. Come up with a scheme that removes unnecessary null-dereference checks for Java programs when the code of all the classes, though unavailable statically, is eventually available during JIT compilation. The precision should be more than an intraprocedural analysis, but you should not have to perform a costly interprocedural analysis in the JIT compiler. Also mention the outcome of your scheme for the above example.

Q5: PARALLEL THANDAI [2+3]

(A) In the class we had seen examples of three compiler passes that may become invalid in presence of multiple threads — loop-invariant code motion, common subexpression elimination, register allocation — and the way(s) we could enable the same by adding some restrictions and/or addons. Construct another example of an optimization (other than those three passes) that gets affected in a similar sense, and explain how could you enable the same without affecting the program semantics.

(B) In the following trace of a concurrent program, each `Ti` is a thread, `x` is a variable, and `v1`, `v2` and `v3` are different values written to or read from `x`. As an example, `R(x)_v1` written against T2 indicates that the thread T2 read the value of `x` as `v1`. The time grows in horizontal steps.

```
T1:   W(x)_v1                          W(x)_v3
T2:             R(x)_v1   W(x)_v2
T3:             R(x)_v1                        R(x)_v3   R(x)_v2
T4:             R(x)_v1                        R(x)_v2   R(x)_v3
```

Explain which of the following memory models would allow or disallow the behaviour depicted by this trace, and how: (i) sequential consistency; (ii) total store ordering; (iii) partial store ordering.

Q6: BONUSI PAAN [1]

Most of you expressed that you learnt a lot in this course. The teaching team too tried to create a good balance between theory and practice, as well as induced some research. If you get a chance, which problem would you like to explore more and expand the limits of, from the ones learnt in CS614.

--*-*- NEXT SHOW: JAN 2024 :-) -*-*-*-*