CS243 Final Examination

Winter 2018

March 23, 2018

This is an open-book, open-notes, open-laptop, closed-network exam. Please do not post anything on Piazza until the solutions are put up on the class website.

You have 3 hours to work on this exam. The examination has 7 problems worth 160 points. Please budget your time accordingly. Write your answers in the space provided on the exam. If you use additional scratch paper, please turn that in as well.

Your Name: ___________________ SUNet ID: ___________________

The following is a statement of the Stanford University Honor Code:

a. The Honor Code is an undertaking of the students, individually and collectively:

   (i) that they will not give or receive aid in examinations; that they will not give or receive unpermitted aid in class work, in the preparation of reports, or in any other work that is to be used by the instructor as the basis of grading;

   (ii) that they will do their share and take an active part in seeing to it that others as well as themselves uphold the spirit and letter of the Honor Code.

b. The faculty on its part manifests its confidence in the honor of its students by refraining from proctoring examinations and from taking unusual and unreasonable precautions to prevent the forms of dishonesty mentioned above. The faculty will also avoid, as far as practicable, academic procedures that create temptations to violate the Honor Code.

c. While the faculty alone has the right and obligation to set academic requirements, the students and faculty will work together to establish optimal conditions for honorable academic work.

Signature: __________________________

<table>
<thead>
<tr>
<th>Problem</th>
<th>#1</th>
<th>#2</th>
<th>#3</th>
<th>#4</th>
<th>#5</th>
<th>#6</th>
<th>#7</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Score</td>
<td>20</td>
<td>20</td>
<td>25</td>
<td>25</td>
<td>20</td>
<td>30</td>
<td>20</td>
<td>160</td>
</tr>
</tbody>
</table>
Problem 1. True or False? Briefly justify your answer in 1 to 5 lines [20 Points]

a. Partial Redundancy Elimination can change the order in which exceptions (or runtime signals such as segmentation fault) are thrown in a program.

True. All PRE needs before it moves an expression is that it is definitely executed (anticipated). PRE can also change the relative order of expressions and instructions. In that case, it might reverse the order of e.g. a division by zero and a null pointer dereference, causing the program to see different runtime behavior. In some languages, such as C, C++, (unsafe) Rust and others, this is acceptable because such operations are undefined. In some other languages, such as Java, where exceptions are well defined, the compiler needs extra care not to switch dangerous expressions around.

b. When applied to a program whose max loop depth is $d$, a forward monotone data flow analysis that visits all nodes in reverse postorder is guaranteed to converge in at most $d + 2$ iterations.

False. Only if data propagates along acyclic paths (sec 9.6.7 of textbook).

Solutions that pointed out we forgot the requirement of “finite descending chains” also got full marks.

c. Which of the following loop transformations can be expressed as an affine transformation of the iteration space and iteration indices?

(i) Interchange True (the matrix is a permutation matrix)

(ii) Fusion True

(iii) Fission True
(iv) Blocking False

d. A generational garbage collector has lower GC total time than a semi-space copying collector for all possible programs.

False, because we can construct a program that allocates objects which don’t die until you reach half the memory.

e. If the maximum number of live ranges at any program point of a program is less than or equal to \( n \), the program can be register allocated with \( n \) registers without spilling

False. Consider this case:

\[
x = \ldots \\
\text{while}(\ldots) \{ \\
\quad y = \ldots \\
\quad \ldots = x \\
\quad z = \ldots \\
\quad \ldots = y \\
\quad x = \ldots \\
\quad \ldots = z \\
\}
\]

There are three live ranges, and each point only two of them intersect, but you need 3 registers to color this graph.
Problem 2. Characteristics of Program Analyses [20 Points]

For each of these analyses, indicate if the known algorithm runs in polynomial or exponential time in the worst case, whether false positive are possible, and whether false negatives are possible. Justify your answers.

a. Liveness analysis

Running time: average $O(dm + n)$ in SSA form, $O(dnm)$ otherwise, where $n$ are the variables and $m$ the instructions (assuming $O(1)$ average time for set operations) and $d$ is the maximum loop depth. It is enough to say “polynomial”, with justification, to receive full credit. False positives: yes, false negatives: no (a variable can be marked live even if the path that uses the variable is impossible, but if the variable is used it will be marked live).

b. Data dependence analysis for affine array accesses

Running time: exponential (due to the integer linear programming). False positives: no, the analysis is exact, assuming only affine array accesses are present, with loops of affine bound and affine conditions on the if-statement; partial/full credit to other answers if motivated. False negatives: never, the analysis is conservative (no credit if you said yes).

c. The context-sensitive pointer analysis presented in class

Running time: exponential (because the number of contexts can be exponential, and because the datalog operations can be exponential). False positives: yes (it can produce more $hP$ tuples than are in the real program), false negatives: no (if a real object allocated at $h1$ points to a real object allocated at $h2$ in some execution of the program, the tuple $hP(h1, x, h2)$ will be generated).

d. The Satisfiability Modulo Theory analysis presented in class for a program with loops, integer and array operations, but no memory or IO operations. Running time: exponential (due to SAT). False positives: no, if the SMT reports satisfiable it produces a model corresponding to a bug. False negatives: yes, because of unrolling.
Problem 3. Partial Redundancy Elimination [25 Points]

Show the result of running partial redundancy elimination on the following program. What’s the final optimized flow graph? You don’t need to show the intermediate steps.

Solution:
Problem 4. Type Analysis [25 Points]

Your task is to use data flow analysis to optimize programs in MiniJ. MiniJ, a super-small subset of Javascript, is dynamically typed. A variable can hold values of different types at different times. If a variable is proven to have a single type, the compiler can simplify the representation and the execution. For example, if a variable is determined to be only a string statically, the compiler can replace the generic + operation with a string concatenation.

Your language has the following types: number, string, object, null, and the following operations:

- \( x = const \): initializes \( x \) to a constant;
- \( x = y \): copies \( y \) into \( x \);
- \( z = x + y \): if either \( x \) or \( y \) is a string, converts the other argument into a string and concatenates the strings; otherwise converts both arguments to a number and returns the sum;
- \( x = y.length \): if \( y \) is a string, sets \( x \) to its length (a number); if \( y \) is an object, reads the “length” property, which could be any type; otherwise sets \( x \) to “null”;
- \( x = y.z \): if \( y \) is an object, reads the \( z \) property from it, which could be any type; otherwise sets \( x \) to “null” (\( z \) is an identifier and is a compile-time constant, not a variable);
- if, else, while, break, continue, return: these have the usual meaning and affect only the control flow;

Assume that variables are preinitialized to some type, including objects and their properties. Note that for this problem, you don’t need to consider property assignments.

The input to your program will be a control flow graph, representing a single procedure in your language. The output should be an assignment from each variable, at each program point, to a set of possible types (not just whether a variable has a single type or not). Your analysis should be precise enough to reason that, in this example, the first + operator is string concatenation, and the second is addition.

```plaintext
x = "hello"
if ... {
    y = "world"
    x = x + y
    x = x.length
} else {
    x = 0
}

y = null
x = x + y
```

The solution is forward dataflow analysis, with a semi-lattice over the powerset of \{ number, string, object, null \}; the meet operator is union. Initialize the types of all variables initially to top (empty set). The transfer functions are defined as:
• $x = \text{const}: \text{out}[x] = \{ \text{type of const} \}$

• $x = y$: $\text{out}[x] = \text{in}[y]$

• $z = x + y$: $\text{out}[z] = \{ \text{string if string} \in \text{in}[x] \text{ or string} \in \text{in}[y] ; \text{number if in}[x] \text{ or in}[y] \text{ contain any other type} \}$

• $x = y.lenght$: $\text{out}[x] = \{ \text{number if string} \in \text{in}[y] ; \text{any type if object} \in \text{in}[y] ; \text{null if in}[y] \text{ contains any other type} \}$

• $x = y.z$: $\text{out}[x] = \{ \text{any type if object} \in \text{in}[y] ; \text{null if in}[y] \text{ contains any other type} \}$

A variant of the solution would be to set all variables initially to $\{ \text{null} \}$ (matching the semantics of common dynamically typed languages). This was also acceptable, provided it was made explicit.

Rubric: 2 pts to choose dataflow over something else, 3 pts for forward pass, 10 points to the semi-lattice, 10 points to the transfer function (2 pt each) Partial credit (1/2 pt) to transfer functions that misinterpreted what “conversion” means and had it affect the inputs as well. Partial credit (6/10 pt) to semi-lattice where the “set of possible types” was not correctly unioned and was treated as three-state (empty / singleton / full). Same for semi-lattice that used three variable state without calling it “set of possible types”.
Problem 5. Software pipelining [20 Points]

Consider the following data dependency graph. Next to each instruction are indicated the resources used by that instruction.

a. What is the minimum initiation interval from the precedence constraints?

There is a cycle of length 2 with an iteration delay of 1, so the minimum interval from precedence would be 2.
2 points.

b. What is the minimum initiation interval from resource usage?

We need 3 of the middle resource, so the minimum interval is 3.
2 points.
c. Construct the optimal software pipelined schedule for this loop. Write both the modulo reservation table and the generated code (with \texttt{nop}s if necessary) for a single iteration of the loop.

Modulo reservation table:

\[
\begin{array}{ccc}
D & A \\
D & C & C \\
B & B & \\
\end{array}
\]

Generated code for a single iteration of the loop:

A
\texttt{nop}
B
\texttt{nop}
C
\texttt{nop}
D
\texttt{nop}

Rubric: 8 points (4 for the opt soft pipeline schedule, and 4 for the generated code)
Partial credit if you omitted one or more \texttt{nop}.

A partial credit (3/4 pt) modulo table is:

\[
\begin{array}{ccc}
\text{B} & \text{B} & \text{A} \\
\text{D} \\
\text{D} & \text{C} & \text{C} \\
\end{array}
\]

This is a partial credit solution because it has the same throughput but slightly worse latency.

Generated code for a single iteration of the loop with that table:

A
\texttt{nop}
\texttt{nop}
B
\texttt{nop}
C
\texttt{nop}
D
\texttt{nop}

Solutions with 4 or more cycles in the initiation interval received less partial credit.
The rest of the problem was graded based on what you answered here.

d. Can this schedule be generated using the heuristic algorithm described in class?

No: the algorithm will schedule A first, then the strongly connected component B-C in the next 2 slots, and now there is no space to fit D.

4 points.
e. Now consider the optimal schedule again. What is the latency of a single iteration? What is the throughput (number of iterations per cycle)?

Latency: 8 clocks. Throughput: 1 iteration in 3 cycles = 0.33 iterations/cycle.

2 points.

f. How does the software pipelined solution compare to just basic block scheduling of the loop body?

Without software pipelining you would schedule in 5 clocks (with a latency of 6), with a throughput of 0.2 iterations/cycle. With software pipelining you get worse latency but better throughput.

2 points.
Problem 6. Parallelization [30 points]
Consider the following program.

```java
for (int i = 1; i < n; i++) {
    for (int j = 1; j < n; j++) {
        A[i, j] = c * A[i, j-1];
    }
}
for (int i = 1; i < n-1; i++) {
    for (int j = 1; j < n; j++) {
    }
}
```

Assume $A$ and $B$ are two non-overlapping $n \times n$ matrices. Both matrices are stored in row-major layout.

a. Draw the iteration space for the program. Use arrows to mark data-dependencies between iterations.

Two square diagrams, one where the arrows point horizontally and one where they point vertically. In both diagrams all the arrows point in one direction (there are no data dependencies on $A$ in the second loop).

It is also possible, and a little more complicated, to draw the diagram for both loops at the same time. Then the result is a 3d diagram with two layers for the two loops, and the arrows point in multiple directions.

2 points

b. Parallelize each loop nest individually. Show the transformed code, and describe any transformation you performed.

The two loops are individually parallelizable: the first one is ok as is (and you assign each row to a different processor), the second one is parallelizable after you interchange $i$ and $j$ (assigning each column to a different processor).

8 points

Solutions that rewrote first loop to do powers of $c$ (which are not correct because floating point math is not associative) received partial credit.
c. Can you parallelize this program with no synchronization? Show the transformed code, or provide a justification that it is not possible.

There is no communication-free parallelism in this code: the solution to the affine mapping constraint has 0 rank.

4 points

d. Can this program be written as a single loop nest with pipelined parallelism? Show the fully permutable loop nest and the best generated parallel code (using any necessary synchronization primitive, as in Homework 6), or provide a justification that it is not. You do not need to perform blocking.

Fusion is possible, with a skewing of the second loop. The fully permutable loop nest looks like:

```cpp
for (int i = 1; i < n; i++) {
    for (int j = 1; j < n; j++) {
        A[i, j] = c * A[i, j-1];
        if (i > 1)
    }
}
```

The best parallel code therefore is:

```cpp
syncvar s[N-1];
for (int i = 1; i < n; i++) {
    for (int j = 1; j < n; j++) {
        if (i > 1)
            s[i-2].wait_until(j);
        A[i, j] = c * A[i, j-1];
        if (i > 1)
        s[i-1].increment();
    }
}
```

This solution assigns each row to a different processor; processor i waits until processor i-1 has completed column j.

12 points.

Alternatively, you can skew the first loop and the write to A. This solution also receives 10 or 12 pts depending on the loop order.
Alternatively, you can move the synchronization to middle of the loop, which changes what offsets you synchronize on. This solution also receives full credit, even though it generates a loop nest that is not fully permutable.

A slightly different “solution” had an if-statement guarding the assignment to A, and different loop bounds. This would have been the correct solution to the original incorrect problem statement (ignoring the out of bound reads), therefore it receives full points.

An alternative solution switches the i and j loops (because they are permutable). This is suboptimal, because the matrices are row major and the solution assigns a column to each processor. This alternative receives 10/12 points.

If you fuse the loops naively, the result is incorrect: you read from $A[i+1,j]$ before the value has been computed. A fused solution without skewing receives 6/12 points. Solutions that apply skewing but compute incorrect answers due to minor mistakes (e.g. off by one) receive 10/12 if the loop order is correct, 8/10 otherwise. Solutions that compute completely incorrect answers receive 4/12 pts. If you omit the write to A, you lose 2 points, regardless of the rest of the solution.

If you do not show the generated parallel code, but your solution is correct, your grade is capped to 8/12. Conversely, if you show the parallel code but not the permutable loop nest, you can still receive full score.

e. Which of the original code, or any of the parallel versions, is likely to perform best? Assume any necessary blocking is now performed.

Parallelizing individual loops is bad from a cache point of view, because in the first loop each thread is assigned a column of the matrix, which means many thread end up writing to the same cache line, causing a lot of thrashing. Even with blocking, you need to load and store the entire A matrix first, then load all of A again when processing B.

In the second loop, loads from A to write to B are cached (because A was loaded immediately before) and fast.

Solutions that convincingly argued otherwise received 2/4 pts.

4 points.
Problem 7. Pointer Analysis [20 points]

Perform context-sensitive pointer analysis on the following code. Draw the cloned call graph starting at main, and write down the list of hP tuples in the end. You do not need to show intermediate steps.

class A {
    static void foo(A a1, B b1) {
        a1.x = b1;
        b1.y = a1;

        B b2 = a1.x;
        b2.y = a1;
    }

    static void bar(B b3) {
        A a2 = new A; // h1
        B b4 = new B; // h2
        foo(a2, b4);

        a2.x = b3;
    }

    static void main() {
        A a3 = new A; // h3
        B b5 = new B; // h4

        foo(a3, b5);

        B b6 = new B; // h5
        bar(b6);
    }
}

The cloned callgraph has main() pointing to foo1() and to bar(); bar() points to foo2(). The list of tuples in the end:

- hP(h3, x, h4)
- hP(h4, y, h3)
- hP(h1, x, h5)
- hP(h1, x, h2)
• hP(h2, y, h1)

• hP(h5, y, h1)

The last tuple, which many missed, exists because b2 is set to all possible values that h1 can point to through x, so vP(b2, h2) and vP(b2, h5). Hence, hP(h5, y, h1).

In a context-insensitive analysis, you would also have hP(h1, x, h4), hP(h3, x, h5) etc. (any A object points to any B object through x, and any B object points to any A object through y). In a flow-sensitive analysis, you would not have hP(h5, y, h1), because a1.x can only point to h2 at that point (it was just set).

Rubric: 2 pts to each node in the call graph. 7 pts for the tuples except hP(h5, y, h1), and 3 pt for the last tuple. -1 for each extra tuple.