CS 243 Homework 5

Winter 2024

Due: February 28, 2024 at 11:59pm

Directions:

- This homework includes Gradiance quizzes and some written questions. The written part should be submitted via Gradescope. No late days can be applied to the Gradiance quiz.
- You may use up to two of your remaining late days for this assignment.
- This assignment is an individual assignment. You are allowed to discuss the homework with others, but you must write the solution individually. If you look up any material in the textbook or online, you should cite it appropriately.

Problem 1. Short answer questions:

1. One way to schedule a do-all loop is fully unroll it and utilize instruction scheduling. The other way is to use software pipelining. For both the techniques, when would you prefer to use one technique over the other and why?

2. Consider the following software pipelining algorithm for do-all loops:

```
1. Find lower bound of initiation interval
2. For T = T0, T0+1, ... until a schedule is found
3.     For each node n in topological order
4.         s0 = earliest n can be scheduled
5.         For each s = s0, s0+1,..., s0+T-1
6.             if NodeScheduled(n, s) break;
7.         if (n cannot be scheduled) break;
8.
9. NodeScheduled(n, s):
10.  Check resources of n at s in modulo resource reservation table
```

   (a) What is the maximum `T` in line 2 that we may need to iterate over?

   (b) If we replace line 2 with a binary search over `T0 to T_max` (from your answer above), does the algorithm produce the optimal schedule (regardless of the time it takes to find the optimal schedule)?

   (c) In line 5, if we replace the upper bound of the loop with `s0` + `T`, can it potentially find an optimal schedule for the initiation interval in cases where it was not able to before?

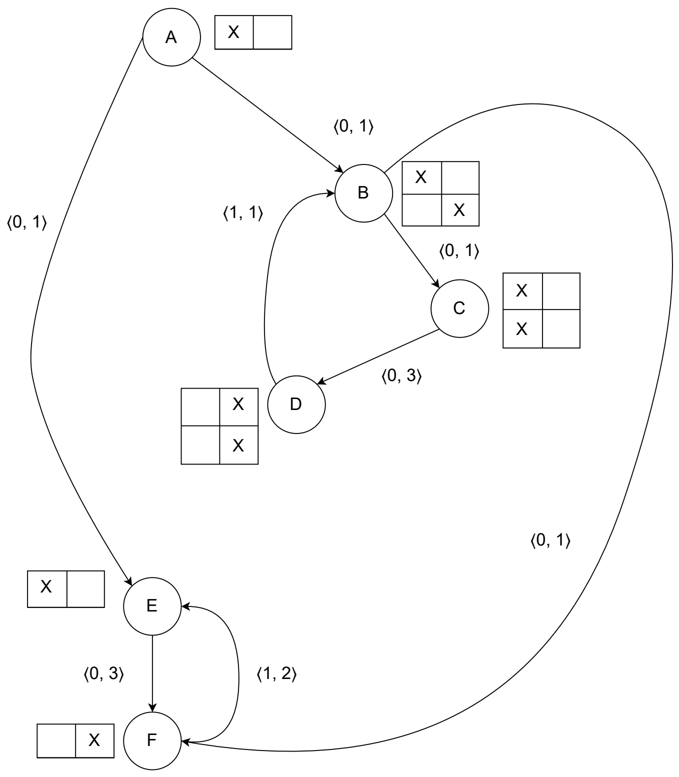
   (d) If we replace line 5 with a binary search similar to the one discussed in part (b) above, does the new algorithm produce the optimal schedule (regardless of the time it takes to find the optimal schedule)?

3. After successfully software pipelining a loop with an initiation interval T, registers `R1` and `R2` in the original program have live ranges of $3 \times T$ and $4 \times T$ cycles, respectively. Suppose you have 10 registers to devote to `R1` and `R2`, how many times would you unroll the steady state of the loop?

4. True or False. A single-issue machine will not benefit from global instruction scheduling because it cannot issue more than one instruction per cycle.

Problem 2. Software Pipelining

Consider the following dependence graph for a single iteration of a loop, with resource constraints:



1. What is the bound on the initiation interval T according to the precedence and resource constraints for this program?

2. What is the minimum initiation interval? Show a modulo reservation table for an optimal software pipelined schedule. Also show the code and schedule for an iteration in the source loop.

3. Can the scheduling algorithm described in class produce the optimal schedule for this loop? If not, show the modulo reservation table and code schedule generated by the algorithm.

Problem 3. Software Pipelining with Register Allocation

Consider the following do-all loop program:

```
for (i = 0; i < 1000; i++)
    C[i] = (A[i] - b) * A[i];
```

One iteration of the loop can be written in assembly as:

```
1.  LD   R5, 0(R1++)     // R1 = &A[i]; R5 = *R1; R1++
2.  SUB  R6, R5, R2      // R2 = b; R6 = R5 - R2
3.
4.  MUL  R6, R6, R5      // R6 = R6 * R5
5.
6.  ST   R6, 0(R3++)     // R3 = &C[i]; *R3 = R6; R3++
```

Optimize this program using software pipelining for a machine with the following specifications:

- The processor can issue at most one `LD` instruction, one `ST` instruction, *and* one arithmetic instruction within the same clock cycle.
- Each arithmetic operation has a two-cycle latency, but can be pipelined.
- The processor supports auto-incrementing addressing and hardware loop operations.

Answer the questions below using your optimized program:

1. What is the minimum initiation interval?
2. Write down the pipelined portion of the loop using actual registers. Annotate each instruction with the iteration it's associated with (e.g., i, $i + 1$, $i + 2$, etc.). Resolve any anti-dependencies across loop iterations by unrolling the loop as described in class.
3. How many more registers does software pipelining require compared to the unpipelined code? Think of a way we can reduce the number of registers used without decreasing the throughput. (You do not have to show the generated code.) *Hint*: check the live ranges for all the duplicated registers.

Problem 4. Dependency Analysis and Parallelization

Consider the C-style program:

```
for (i = k+2; i < n; i++) {
    for (j = 2*i+m; j < 6*n-i; j++) {
        X[i,   4*j-2] = X[i, 4*j+1] + Y[i, j]
        Y[i+1,   j-3] = X[i, 2*j]   + Y[i, j]
    }
}
```

Assuming $0 \leq m \leq k \ll n$ and that X and Y are non-overlapping arrays, answer the following questions.

1. Draw the iteration space for this loop. *Hint*: The axes should be situated such that program execution first goes from left to right, and next from bottom to top.
2. What are the data dependencies in this loop? Categorize each as a true dependency, output dependency, or an anti-dependency. Include all dependency, even if they can be ruled out using simple tests like the GCD test. *Hint*: a data dependency is something of the form "read/write $A[i]$, read/write $B[j]$."
3. Formulate the data dependence tests for the given loop nest.
4. Which pairs of accesses are data dependent?

Problem 5. Affine Transforms

Apply affine transform to find the largest degree of outermost loop parallelism. Show the transformed code, and mark the loops that are parallelizable. Assume $N \geq 100$.

```
for (i = 1; i <= N; i++) {
    for (j = 1; j <= N; j++) {
        A[i,j] = A[i-1,j] + X[i,j];
    }
}
for (i = 4; i <= N; i++) {
    for (j = i+1; j <= N; j++) {
        for (k = 1; k <= N; k++) {
            B[i,k] = B[i-1,k] + Y[j];
        }
    }
}
for (i = 1; i <= N; i++) {
    C[i] = A[3,i];
}
```