CS243 Homework 5

Winter 2020

Due: February 26th, 2020 at 4:30 pm

Directions:

• Submit via Gradescope.

• You may use up to two of your remaining late days for this assignment, for a late
deadline of February 28th, 2020 at 4:30 pm.

• Remember to complete the corresponding Gradiance quizzes by the start of class on
the due date. There are no late days for Gradiance.

• This 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.** Software Pipelining

Consider the following loop:

```c
for (i = 1; i < n; i++) {
    X[i+1] = X[i] + Y[i];
    Y[i+3] = X[m] * 2;
}
```

$m$ in the above code is some constant. Assume LD and ST take 1 clock, ADD and MUL take 2 clocks. The machine has two MEM units that can execute one LD and one ST, and two ALU units that can execute ADD and MUL in one clock. The machine can autoincrement address registers.

1. Draw the data dependence graph by showing three types of nodes, LD, ADD, and ST.

2. Label the edges of the dependency graph according to the type of dependency (true dependency, anti-dependency or output dependency).

3. What is the lower bound on the initiation interval?
Problem 2. Dependency Analysis and Parallelization

for (i = 1; i < n; i++) {
    for (j = i+1; j < i+m; j++) {
        X[4*i, j] = X[8*i+1, j] + Y[i, j];
        Y[j-2, i] = X[8*i+3, j] + Y[i, j];
    }
}

1. Draw the iteration space for this loop.

2. What are the data dependencies in this loop? Hint: a data dependency is something of the form “read/write A[i], read/write B[j]”.

3. What are the data dependency equations that must be satisfied for the loop to be parallelizable?

4. Is this loop nest 1 or 2-d parallelizable without loop transformations? You do not need to show the exact solution to the equations, but justify your answer.
Problem 3. Global Instruction Scheduling Assume you have a statically scheduled machine that can only issue one operation every clock. All operations have a latency of one clock cycle, with the exception of its memory load operation, which has a latency of three clock cycles. Consider the following locally scheduled program:

```
L0
w = x;
h = y;
m = w + h;
if (m >= 0) go to L1;
```

```
L2
if (m == 0) go to L3;
```

```
L1
f = m + 2;
g = *f;
nop;
nop;
h = g + g;
```

```
L3
w = *h;
nop;
nop;
h = 2 * w + 1;
```

```
L4
r = h + w;
```

Assume that only r is live at the end of the program. Each branch in the flow graph is labeled with the probability that it is taken dynamically. To answer the following, you may apply any of the code motions discussed in class, but no other optimizations.

1. Is this the best globally scheduled code that can be generated given that p = 0.1? If not, provide the improved code along with its expected execution time.

2. Repeat part 1 given that p = 0.5.

3. What is the percentage improvement in execution time from the original program for your programs in part 1 and 2, if one was provided?
Problem 4. More 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 5.** Affine Transforms

Apply affine transform to find the largest degree of outermost loop parallelism. Show the transformed code, marking the loops that are parallelizable.

```c
for (i = 1; i <= N; i++) {
    for (j = 1; j <= N; j++) {
        A[i,j] = A[i-1,j] + X[i,j];
    }
}
for (i = 2; 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++) {
}
```