CS 243 Homework 5

Winter 2023

Due: March 1, 2023 at 11:59pm

Directions:

• This homework includes a Gradiance quiz 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, for a late deadline of March 3, 2023 at 11:59pm.

• 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. Global Instruction Scheduling

Assume you have a machine with a statically scheduled processor 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:

\[
\begin{align*}
&d = *a; \\
&\text{nop;}
&\text{nop;}
&\text{if } (d \geq 0) \text{ goto } L1; \\
&1 - p
\end{align*}
\]

\[
\begin{align*}
&L0 \\
&L1 \text{ if } (d == 0) \text{ goto } L3; \\
&L2 \text{ } f = *d; \\
&\text{nop;}
&\text{nop;}
&k = f + 8;
\end{align*}
\]

\[
\begin{align*}
&L0 \\
&L1 \text{ if } (d == 0) \text{ goto } L3; \\
&L2 \text{ } f = *d; \\
&\text{nop;}
&\text{nop;}
&k = f + 8;
\end{align*}
\]

\[
\begin{align*}
&L3 \text{ } k = *d; \\
&\text{nop;}
&\text{nop;}
&c = k + k;
\end{align*}
\]

\[
\begin{align*}
&L4 \text{ } w = a + c; \\
&t = k + w; \\
&\text{L4 EXIT}
\end{align*}
\]

All variables in the program are defined before the entry (L0) of the program. Assume that only variable \( t \) is live at the exit of the program. Each branch in the flow graph is labeled with the probability that it is taken dynamically. To answer the following questions, you may apply any of the code motions discussed in class, but no other optimizations.

(See next page for questions.)
1. Is this the best globally scheduled code that can be generated given that \( p = 0.9 \)? If not, provide the improved code along with its expected execution time. **Note:** for an if branch instruction, assume that if the branch corresponding to “goto L” is taken, it costs two cycles; otherwise it costs only one.

2. Repeat part 1, but instead assume that \( p = 0.1 \).
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:

```c
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. MUL R6, R6, R5 // R6 = R6 * R5
4. 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:

```c
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. Is the loop nest 1-d parallelizable or 2-d parallelizable without loop transformations (or not parallelizable at all)? If not parallelizable, provide an example pair of loop indices that cause the conflict.
**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 $N \geq 100$.

```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 = 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];
}
```