Lecture 8
Software Pipelining

I. Introduction
II. DoAll Loops Scheduling
III. DoAcross Loops Scheduling
IV. Register Allocation
Reading: Chapter 10.5 – 10.6
Software Pipelining

• Numerical code has lots of parallelism
  – Often with small kernels and many iterations

• Global scheduling is not effective

• Software pipelining focuses on loops
  – The goal is to find the optimal schedule

Software Pipelining: An Effective Scheduling Technique for VLIW Machines.
M. Lam.
Outline

- Scheduling algorithm, ignoring anti-dependences in registers
  - Can eliminate anti-dependences with better register allocation

- Register allocation
I. Example of DoAll Loops

- **Machine:**
  - Per clock: 1 read, 1 write, 1 (2-stage) arithmetic op, with hardware loop op and auto-incrementing addressing mode.

- **Source code:**
  ```
  For i = 1 to n
  ```

- **Code for one iteration:**
  1. LD  R5,0(R1++)
  2. LD  R6,0(R2++)
  3. MUL R7,R5,R6
  4. 
  5. ADD R8,R7,R4
  6. 
  7. ST 0(R3++),R8

- **No parallelism in basic block**
Unrolling (ignoring anti-dependences for now)

<table>
<thead>
<tr>
<th>i = 1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. L:</td>
<td>LD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td>LD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td>LD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.</td>
<td>MUL</td>
<td>LD</td>
<td></td>
</tr>
<tr>
<td>5.</td>
<td>MUL</td>
<td>LD</td>
<td></td>
</tr>
<tr>
<td>6.</td>
<td>ADD</td>
<td>LD</td>
<td></td>
</tr>
<tr>
<td>7.</td>
<td>ADD</td>
<td></td>
<td>LD</td>
</tr>
<tr>
<td>8.</td>
<td>ST</td>
<td>MUL</td>
<td>LD</td>
</tr>
<tr>
<td>9.</td>
<td></td>
<td>MUL</td>
<td></td>
</tr>
<tr>
<td>10.</td>
<td>ST</td>
<td>ADD</td>
<td></td>
</tr>
<tr>
<td>11.</td>
<td></td>
<td>ADD</td>
<td></td>
</tr>
<tr>
<td>12.</td>
<td></td>
<td>ST</td>
<td></td>
</tr>
<tr>
<td>13.</td>
<td></td>
<td>ST</td>
<td>BL  (L)</td>
</tr>
</tbody>
</table>

• Let \( u \) be the degree of unrolling:
  – Length of \( u \) iterations = \( 7 + 2(u-1) \)
  – Execution time per source iteration = \( \frac{7+2(u-1)}{u} = 2 + \frac{5}{u} \)
Software Pipelined Code

\[ i = 1 \]

1. LD
2. LD
3. MUL LD
4. LD
5. MUL LD
6. ADD LD
7. MUL LD
8. ST ADD LD
9. MUL LD
10. ST ADD LD
11. ST MUL
12. ST ADD
13. 
14. ST ADD
15. 
16. ST

Schedule of 1 iteration

LD Alu ST
Final Software Pipelined Code

1. LD
2. LD
3. MUL    LD
4.        LD
5.        MUL    LD
6. ADD           LD
7.               MUL    LD
8. ST     ADD           LD     BL 7
9.                      MUL
10.        ST     ADD
11.        ST     ADD
12.        ST     ADD
13.        ST
14.        ST

• Unlike unrolling, software pipelining can give optimal result.
• Locally compacted code may not be globally optimal
• DOALL: Can fill arbitrarily long pipelines
Example of DoAcross Loops

Loop:
- Sum = Sum + A[i];
- B[i] = A[i] * c;

Software Pipelined Code:
1. LD
2. MUL
3. ADD
4. ST
5. ADD
6. ST

Doacross loops:
- Recurrences can be parallelized
- Harder to fully utilize hardware with large degrees of parallelism
Problem Formulation

**Goals:**
- maximize throughput
- small code size

**Find:**
- an *identical* relative schedule $S(n)$ for every iteration
- a *constant* initiation interval $(T)$

such that
- the initiation interval is minimized

**Complexity:**
- NP-complete in general

S

0 LD
1 MUL
2 ADD LD
3 ST MUL

ADD

ST

T=2
II. Resources on Bound on Initiation Interval

- **Example:** Resource usage of 1 iteration;
  Machine can execute 1 LD, 1 ST, 2 ALU per clock

  \[
  \text{LD, LD, MUL, ADD, ST}
  \]

- **Lower bound** on initiation interval?

  for all resource \( i \),
  number of units required by one iteration: \( n_i \)
  number of units in system: \( R_i \)

  Lower bound due to resource constraints:
  \[
  \max_i \left\lceil \frac{n_i}{R_i} \right\rceil
  \]
Scheduling Constraints: Resources

• **RT**: resource reservation table for a single iteration
• **RT\_T**: modulo resource reservation table for initiation interval T

\[
RT_T[i] = \sum_{t \mid (t \mod T = i)} RT[t]
\]
Example: DoAll Loops

Quiz: What is a lower bound on the initiation interval due to resources?
Quiz: What is the best schedule (that maximizes throughput)?
Algorithm for DoAll Loops

Find lower bound of initiation interval: \( T_0 \)
  based on resource constraints

For \( T = T_0, T_0+1, \ldots \) until a schedule is found  // Try higher initiation intervals

  For each node \( n \) in topological order
    \( s_0 = \) earliest \( n \) can be scheduled
    for each \( s = s_0, s_0+1, \ldots, s_0+T-1 \)
      if \( \text{NodeScheduled}(n, s) \) break;
      if \( (n \) cannot be scheduled) break;  // Fail for this initiation interval

\textbf{NodeScheduled}(n, s)  // schedule \( n \) at time \( s \)

  – Check resources of \( n \) at \( s \) in modulo resource reservation table

• Can always meet the lower bound if
  – every operation uses only 1 resource, and
  – the loop is a DoAll loop (no recurrences)
III. Do-Across Loops

for (i = 0; i < n; i++) {
    *(p++) = *(q++) + c
}

Quiz: Minimum initiation interval?

Label edges with $< \delta, d >$ ($\delta =$ iteration difference, $d =$ delay)

$S(n)$: Schedule for $n$ with respect to the beginning of the iteration it is in

Constraint for edge from $n_1$ to $n_2$ labeled $< \delta, d >$

$\delta \times T + S(n_2) - S(n_1) \geq d$

$L D^k$ : LD in iteration $k$

0 $\times T + S(ST) - S(LD) \geq 3$

$S(ST) \geq S(LD) + 3$

$1 \times T + S(LD) - S(ST) \geq 1$

$S(ST) \leq S(LD) + T - 1$

When $T=4$, $S(ST) = S(LD)+3$

When $T=5$, $S(LD)+3 \leq S(ST) \leq S(LD)+4$
Another Example

for (i = 2; i < n; i++) {
}

- Label edges with $< \delta, d>$ ($\delta$ = iteration difference, $d$ = delay)
- $S(n)$: Schedule for $n$ with respect to the beginning of the iteration it is in
- Constraint for edge from $n_1$ to $n_2$ labeled $< \delta, d>$
  \[ \delta \times T + S(n_2) - S(n_1) \geq d \]
- Quiz: Minimum initiation interval?
Minimum Initiation Interval
From Cycles in Precedence Constraints

• Minimum initiation interval (MII) = \( \max_c \frac{\text{CycleLength}(c)}{\text{IterationDifference}(c)} \)
  where \( c \) is a cycle in the graph

• Overall MII = \( \max (\text{MII due to resources, MII due to cycles}) \)

• Definition: Critical cycles are cycles that have the largest \( \frac{\text{CycleLength}(c)}{\text{IterationDifference}(c)} \)

• When \( T = \text{MII due to cycles in precedence constraints} \)
  – Nodes in the critical cycle have no slack in scheduling!
  – The schedule of one node in the cycle, determines the schedule of all nodes in the cycle!

• Increasing the initiation interval increases the slack
Cyclic Graphs

Quiz: MII due to resources?  MII due to cycles?
Strongly Connected Components

- **A strongly connected component (SCC)**
  - Set of nodes such that every node can reach every other node

- **Every node constrains all others from above and below**
  - Finds longest paths between every pair of nodes
  - As each node scheduled, find lower and upper bounds of all other nodes in SCC

- **SCCs are hard to schedule**
  - Critical cycle: no slack
    - Backtrack starting with the first node in SCC
  - increases T, increases slack

- **Edges between SCCs are acyclic**
  - Topological sort of SCCs in the outerloop
  - Schedule SCC with backtracking
Reminder: Algorithm for DoAll Loops

**Find lower bound of initiation interval: T₀**

- based on resource constraints

**For T = T₀, T₀+1, ... until a schedule is found**  // Try higher initiation intervals

  **For each node n in topological order**
  
  \[ s₀ = \text{earliest } n \text{ can be scheduled} \]
  
  for each \( s = s₀, s₀ + 1, ..., s₀ + T-1 \)
  
  - if NodeScheduled(n, s) break;
  
  if (n cannot be scheduled) break;  // Fail for this initiation interval

**NodeScheduled(n, s)**  // schedule n at time s

- Check resources of \( n \) at \( s \) in modulo resource reservation table
Full Algorithm

Find lower bound of initiation interval: \( T_0 \)
- based on resource constraints and precedence constraints

// Outer loop: Topological sort of SCCs with backtracking
For \( T = T_0, T_0+1, \ldots \), until a schedule is found    // Try higher initiation intervals
  \( E^* \) = longest path between each pair of nodes
  For each SCC c in topological order
    \( s_0 \) = Earliest c can be scheduled
    For each \( s = s_0, s_0 +1, \ldots, s_0 +T-1 \)
      // Change start time of c (backtracking)
      If SCCScheduled(c, s) break;
      If (c cannot be scheduled) break;    // Fail for this initiation interval

SCCScheduled(c, s)
  // Schedule SCC c at time s
  Schedule first node at s, return false if fails
  For each remaining node n in c
    \( s_l \) = lower bound on n based on \( E^* \)
    \( s_u \) = upper bound on n based on \( E^* \)
    For each \( s = s_l, s_l +1, \min (s_l +T-1, s_u) \)
      if NodeScheduled(n, s) break;    // Schedule if no resource conflicts
      if n cannot be scheduled return false;
  Return true;
Outline

- Scheduling algorithm, ignoring anti-dependences in registers
  - Can eliminate anti-dependences with better register allocation

- Register allocation
IV. Register Allocation

• Software-pipelined code

1. LD
2. LD
3. MUL LD
4. LD
5. MUL LD
6. ADD LD
7. MUL LD
8. ST ADD LD BL L
9. MUL
10. ST ADD
11. ST ADD
12. ST ADD
13. ST
14. ST

Schedule of one iteration with register allocation

1. LD R5,0 (R1++)
2. LD R6,0 (R2++)
3. MUL R7,R5,R6
4. 
5. 
6. ADD R8,R7,R4
7. 
8. ST 0 (R3++) ,R8

Problem:
R7 is reassigned in multiply in iteration 2 before it is used for addition in iteration 1

Lifetime of R7 (3) ≥ initiation interval (T=2)

Solution:
Remove anti-dependence by using a different register for iteration 2

Quiz: how many registers do we need for the result of the multiplies?
Modulo Variable Expansion

1. LD R5,0 (R1++)
2. LD R6,0 (R1++)
3. MUL R7,R5,R6
4. MUL R7,R5,R6
5. ADD R8,R7,R7
6. ADD R8,R7,R7
7. ADD R8,R7,R7
8. ST 0 (R3++),R8
9. ADD R8,R7,R7
10. MUL R7,R5,R6
11. ADD R8,R7,R7
12. ADD R8,R7,R7
13. ADD R8,R7,R7
14. MUL R7,R5,R6
15. MUL R7,R5,R6
16. MUL R7,R5,R6

L 7. LD R5,0 (R1++)
8. LD R6,0 (R1++)
9. LD R5,0 (R1++)
10. LD R6,0 (R1++)
11. LD R5,0 (R1++)
12. LD R6,0 (R1++)
13. LD R5,0 (R1++)
14. LD R6,0 (R1++)
15. LD R5,0 (R1++)
16. LD R6,0 (R1++)
Algorithm

- **Normally, every iteration uses the same set of registers**
  - introduces artificial anti-dependences for software pipelining
- **Modulo variable expansion algorithm**
  - schedule each iteration ignoring artificial constraints on registers
  - calculate life times of registers
  - degree of unrolling = \( \max_r \left( \frac{\text{lifetime}_r}{T} \right) \)
  - unroll the steady state of software pipelined loop to use different registers
- **Code generation**
  - generate one pipelined loop with only one exit
    (at beginning of steady state)
  - generate one unpipelined loop to handle the rest
  - code generation is the messiest part of the algorithm!
Conclusions

• **Numerical Code**
  – Software pipelining is useful for machines with a lot of parallelism (which includes the stages of pipelining)
  – Compact code
  – Limits to parallelism: dependence cycles, critical resource

• **General Lessons**
  – Problem formulation: Important to identify
    • the need (parallel hardware),
    • the opportunity (numerical codes have independent operations)
  – Designing the right abstraction to address the key constraint
    • modulo scheduling