Software Pipelining by Modulo Scheduling

Philip Sweany
University of North Texas
Overview

- Opportunities for Loop Optimization
- Software Pipelining
- Modulo Scheduling
- Resource and Dependence Constraints
- Scheduling Pipelines
- Limitations
- Enhancements
- Plans
Why Bother?

- Instruction-level parallelism (ILP) requires scheduling
- “90% of execution time spent in loops”
- Loops can exhibit significant parallelism across iterations of a loop.
- Local and global instruction scheduling don’t address parallelism across loop boundaries
Example Code

```c
int a[4][4];
int b[4][4];
int c[4][4];

main()
{
    int i, j, k;

    for(i=0; i<4; i++)
        for(j=0; j<4; j++)
            c[i][j] = 0;

        for(k=0; k<4; k++)
            c[i][j] += a[i][k] * b[k][j];

}```
Machine Model (for example)

- Long instruction word (LIW)
- Separate add and multiply pipes
- Add produces a result in *one* machine cycle
- Multiply requires *two* machine cycles to produce result
- Multiply is pipelined
Local Schedule — 192 cycles

1. \( t = a[i][k] \times b[k][j] \)
2. nop
3. \( c[i][j] = t + c[i][j] \)
A Pipelined Schedule — 144 cycles

Prelude: executed once
1. t[0] = a[0][0] * b[0][0]

Loop Body: executed for k = 1 to 3
1 nop
2 t[k] = a[i][k] * b[0][0] # c[i][j] += t[k-1]

Postlude: executed once
1 nop
2 c[i][j] += t[3]

Note: Can actually get 96 cycles by overlapping two iterations of the loop in a single 2-cycle loop body.
Software Pipelining Methods

Allan et al. (Computing Surveys, 1995) defines two major approaches to software pipelining:

**Kernel Recognition** unrolls the innermost loop an “appropriate” number of times and searches for a repeating pattern. This repeating pattern then becomes the “loop body.”

**Modulo Scheduling** selects a (minimum) schedule for one loop iteration such that, when the schedule is repeated, no constraints are violated. Length of this schedule is called the *initiation interval*, II.
Iterative Modulo Scheduling

- Schedule loop with the smallest possible II
- Build data dependence graph (DDG)
  - Nodes represent operations to be performed
  - Weighted, directed arcs represent precedence constraint between two nodes
- Compute resource constraint, \( resII \), smallest II that contains enough resources (functional units?) to satisfy all the resources required by one loop iteration.
- Compute recurrence constraint, \( recII \), longest dependence loop in the DDG.
- \( \text{min}II = \max(resII, recII) \)
- Try to schedule with \( II = \text{min}II, II = \text{min}II + 1, \text{min}II + 2, \ldots \) until a valid schedule is found
Scheduling a Pipeline

How do you “Try to schedule with II = minII, II = minII + 1, minII + 2, ... until a valid schedule is found”

- Use “slightly” modified local scheduling algorithm (List scheduling, but with slightly modified heuristics for prioritizing DDG nodes.)
  - Set the schedule length to be the II we’re attempting to schedule
  - Attempt to fit the operations to be scheduled II cycles
- If no schedule found “backtrack” and try again in II cycles.
- After “sufficient” number of backtracking attempts, admit failure.
Dependence

• Operation $A$ is dependent upon operation $B$ iff $B$ precedes $A$ in execution and both operations access the same memory location.

• Four basic types of dependence
  – True dependence (RAW)
  – Anti-dependence (WAR)
  – Output dependence (WAW)
  – Input dependence (RAR)

• Input dependence usually ignored in scheduling
Dependence (cont.)

- Represented in DDG
- Two groups of dependences
  - *Loop-independent* dependence
  - *Loop-carried* dependence
- DDG arcs decorated with two integers, (dist, min)
  - *Dist* is the difference in iterations between the head and tail of the dependence
  - *Min* is the latency required between two nodes
- Dependence analysis is quite complicated, but techniques do exist.
Start of Distance Table — II = 1

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-∞</td>
<td>1</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
</tr>
<tr>
<td>2</td>
<td>-∞</td>
<td>-∞</td>
<td>0</td>
<td>-∞</td>
<td>-∞</td>
<td>1</td>
<td>-∞</td>
</tr>
<tr>
<td>3</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
<td>1</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
</tr>
<tr>
<td>4</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
<td>1</td>
<td>-∞</td>
<td>-∞</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
</tr>
<tr>
<td>6</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
<td>-1</td>
<td>-∞</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>6</td>
<td>$-\infty$</td>
<td>$-\infty$</td>
<td>$-\infty$</td>
<td>$-\infty$</td>
<td>$-\infty$</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>$-\infty$</td>
<td>$-\infty$</td>
<td>$-\infty$</td>
<td>$-\infty$</td>
<td>$-\infty$</td>
<td>-1</td>
<td>0</td>
</tr>
</tbody>
</table>

Distance Table — $\Pi = 1$
<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-∞</td>
<td>1</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
</tr>
<tr>
<td>2</td>
<td>-∞</td>
<td>-∞</td>
<td>-1</td>
<td>-∞</td>
<td>-∞</td>
<td>1</td>
<td>-∞</td>
</tr>
<tr>
<td>3</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
<td>0</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
</tr>
<tr>
<td>4</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
<td>1</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
</tr>
<tr>
<td>5</td>
<td>-1</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
</tr>
<tr>
<td>6</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
<td>1</td>
<td>-∞</td>
</tr>
<tr>
<td>7</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
<td>-3</td>
<td>-∞</td>
</tr>
</tbody>
</table>

Start of Distance Table — II = 2
<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
<td>-2</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
<td>-3</td>
<td>-2</td>
</tr>
</tbody>
</table>

Distance Table — II = 2
<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-∞</td>
<td>1</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
</tr>
<tr>
<td>2</td>
<td>-∞</td>
<td>-∞</td>
<td>-2</td>
<td>-∞</td>
<td>-∞</td>
<td>1</td>
<td>-∞</td>
</tr>
<tr>
<td>3</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
<td>0</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
</tr>
<tr>
<td>4</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
<td>-1</td>
<td>-∞</td>
<td>-∞</td>
</tr>
<tr>
<td>5</td>
<td>-2</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
</tr>
<tr>
<td>6</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
<td>-5</td>
<td>-∞</td>
</tr>
</tbody>
</table>

Start of Distance Table — II = 3
<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-4</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>-2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>-5</td>
<td>-4</td>
<td>-2</td>
<td>-2</td>
<td>-3</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>-3</td>
<td>-2</td>
<td>-4</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>-3</td>
<td>-2</td>
<td>-4</td>
<td>-4</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>-2</td>
<td>-1</td>
<td>-3</td>
<td>-3</td>
<td>-4</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
<td>-4</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
<td>-∞</td>
<td>-5</td>
<td>-4</td>
</tr>
</tbody>
</table>

Distance Table — II = 3
Limitations of Modulo Scheduling

- Control flow makes things difficult
  - Modulo scheduling requires a single DDG
  - Can use if-conversion (Warter) but graph “explodes” quickly
  - Lam’s hierarchical reduction is another method, but not without problems

- Software pipelining in general increases register needs dramatically. Modulo scheduling exacerbates this problem.
  - Techniques exist to minimize required registers but cannot guarantee fitting in machine’s registers
  - To address register problems, some increase II; others spill

- Compile time is a problem, particularly computing the distance matrix
Improved (?) Modulo Scheduling

- Unroll-and-Jam on nested loops can significantly shorten the execution time needed for a loop at the cost of code size and register usage.
- Use of a cache-reuse model can give better schedules than assuming all cache accesses are hits and can reduce register requirements over assuming all accesses are cache misses.
- A plan to software pipeline with a fixed number of registers; like spilling while scheduling.
Unroll-and-Jam

- Requires nested loops
- Unroll outer loop and jam resulting inner loops back together
- Introduces more parallelism into inner loop and it will not lengthen any recurrence cycles.
- Using unroll-and-jam on 26 FORTRAN nested loops before performing modulo scheduling led to:
  - Decreased execution time for loops of up to 94.2%. On average, loops decreased execution time by 56.9%
  - Increased register requirements significantly, often by a factor of 5.
Example Code with Unroll-and-Jam

for(i=0; i<4; i++)
    for(j=0; j<4; j+=2)
    {
        c[i][j] = 0;
        c[i][j+1] = 0;
        for(k=0; k<4; k++)
        {
            c[i][j] += a[i][k] * b[k][j];
            c[i][j+1] += a[i][k] * b[k][j+1];
        }
    }
Using Cache Reuse Information

- Caches lead to uncertain (for compiler) latencies
- Local scheduling typically assumes all cache accesses are hits
- Assuming all hits leads to stalls and slower execution
- Typical modulo schedulers assume all cache accesses are misses. (Modulo scheduling can effectively hide long latency of miss)
- Assuming cache miss stretches register lifetimes and thus leads to greater requirements
Cache Reuse Experiment

Using a simple cache reuse model, our modulo scheduler

- Improved execution time roughly 11% over an all-hit assumption with little change in register usage
- Used 17.9% fewer registers than an all-miss assumption, while generating 8% slower code
Summary

- Modulo scheduling can lead to dramatic speed improvements for single-block loops
- Modulo scheduling does have limitations, namely difficulty with control flow and (potentially) massive register requirements
- At MTU we discovered some enhancements to modulo scheduling to make it even more effective and practical, and in one case improved upon the optimal schedule.