Unrolling-Based Optimizations for Modulo Scheduling

Daniel M. Lavery
Wen-mei W. Hwu

University of Illinois at Urbana-Champaign

Outline

- Modulo scheduling concepts
- Motivation for unrolling
- Unrolling-based optimization case study
- Related work
- SPEC CFP92 experimental results
- Summary and future work

Modulo Scheduling

- Simplifies the process of software pipelining by:
  - using a common schedule for all iterations
  - initiating iterations at a constant rate
    - Initiation Interval (II)
- Determine lower bound on II (MII)
- Attempt to generate a schedule at that II
- There are two constraints on the MII
  - resource requirements
  - dependence cycles

Resources

- The modulo constraint
  - result of the periodic initiation of the same schedule
  - no resource may be used more than once at the same time modulo II
- Enforced using a Modulo Resource Table (MRT)
  - II rows and one column per resource
  - schedule a single iteration
- The MRT must contain a sufficient amount of each resource
  - Resource-constrained lower bound on II (ResMII)

Resource-constrained MII

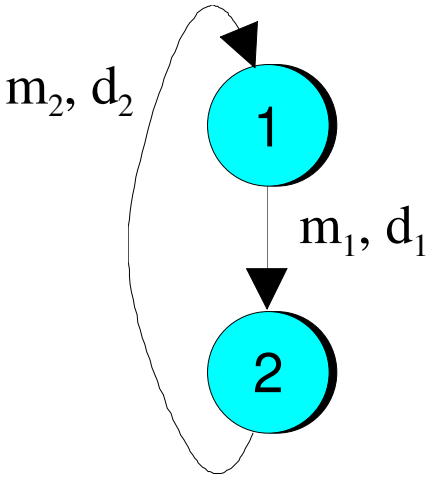
Assembly Code Loop

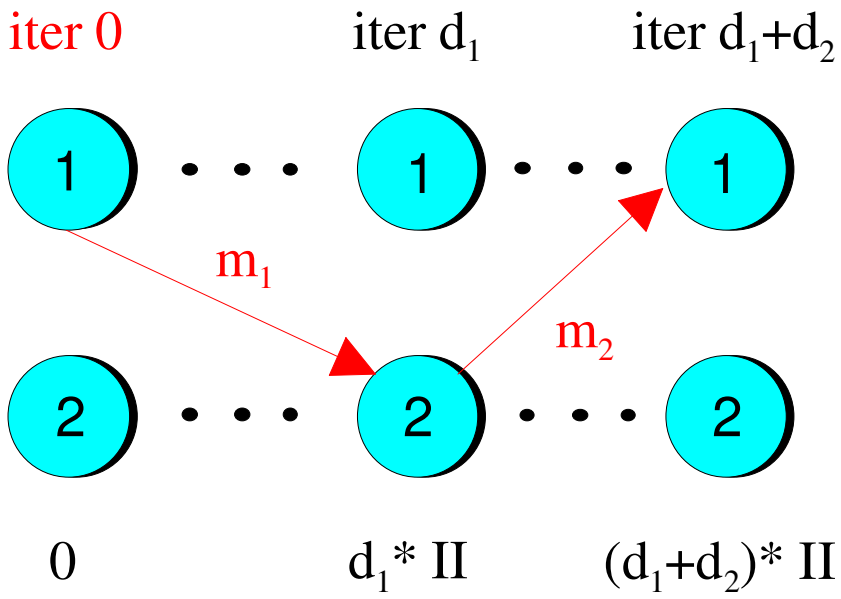
```
L1: f3 = MEM(r8+r4)
    f5 = MEM(r2+r4)
    f6 = f3 * f5
    MEM(r9+r4) = f6
    r4 = r4 +4
    ble (r4 r7) L1
```

MRT

ResMII = 3

Recurrence-constrained MII





Delay(c) = $m_1 + m_2$

Distance(c) = $d_1 + d_2$

Delay(c) $\leq$ Distance(c) * II

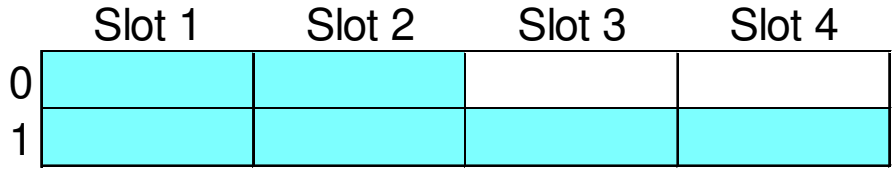
Motivation for Unrolling

- II is restricted to be an integer
  - reduce performance degradation by unrolling
  - retire multiple iterations per cycle
- Comparison to acyclic scheduling of unrolled loops
  - effective II not restricted to be an integer
  - new opportunities for optimization [Mahlke et al. 92]

Degradation Due to Integer II

Same Assembly Loop as Before

Without unrolling

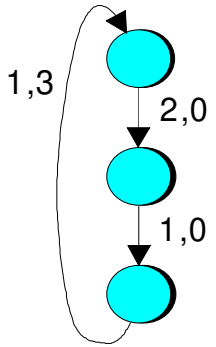


ResMII = 2

Unrolled twice

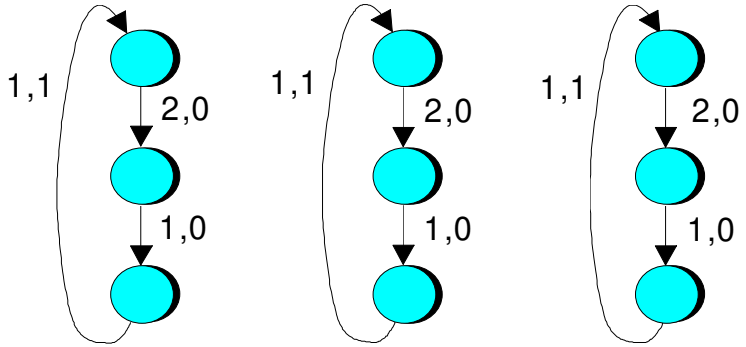
ResMII_eff = 1.5

Without Unrolling



RecMII = 2

Unrolled Three Times



RecMII_eff = 1

Motivation for Unrolling

- II is restricted to be an integer
  - reduce performance degradation by unrolling
  - retire multiple iterations per cycle
- Comparison to acyclic scheduling of unrolled loops
  - effective II not restricted to be an integer
  - new opportunities for optimization [Mahlke et al. 92]

Case Study: Vector-Matrix Product

Source Code

```
do i = 1,n
      C(i) = 0.0
      do j = 1,m
            C(i) = C(i) +A(j) * B(j,i)
      end do
end do
```

Assembly Code for Inner Loop

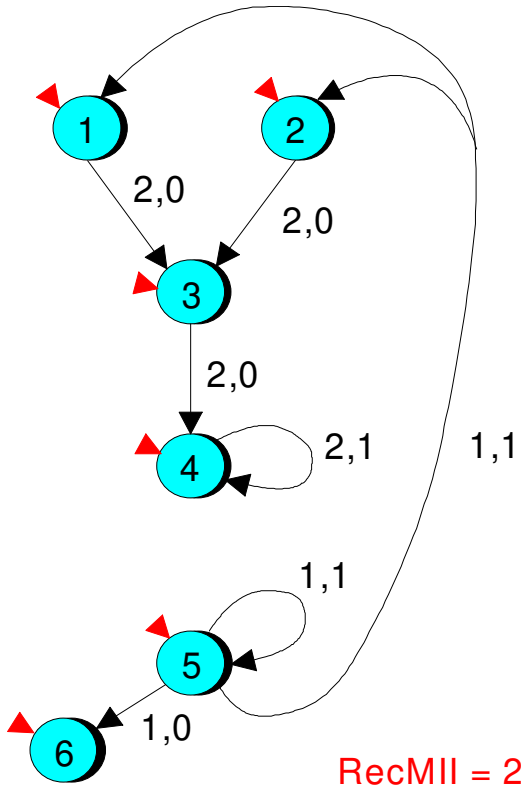
Inst.	Assembly
1	L1: f3 = MEM(r8+r4)
2	f5 = MEM(r2+r4)
3	f6 = f3 * f5
4	f1 = f1 + f6
5	r4 = r4 + 4
6	ble (r4 r7) L1

ResMII = 1

Register Contents

f1 = C(i)
f3 = A(j)
f5 = B(j,i)
r8 = &A(1)
r2 = &B(1,i)
r4 = 4*j
r7 = 4*m

Dependence Graph for Inner Loop



Unrolling and Optimization

Unrolled Three Times

```
      r3 = r7 - 8
      bgt (r4 r3)  L2
L1:   f3 = MEM(r8+r4)
      f5 = MEM(r2+r4)
      f6 = f3 * f5
      f1 = f1 + f6
      r4 = r4 + 4
      f3 = MEM(r8+r4)
      f5 = MEM(r2+r4)
      f6 = f3 * f5
      f1 = f1 + f6
      r4 = r4 + 4
      f3 = MEM(r8+r4)
      f5 = MEM(r2+r4)
      f6 = f3 * f5
      f1 = f1 + f6
      r4 = r4 + 4
      ble (r4 r3)  L1
L2:   Original Loop
```

ResMII_eff: 1 to 0.66
RecMII_eff = Larger

After Accumulator Expansion

```
      f11 = 0.0
      f12 = 0.0
      f13 = 0.0
L1:   f3 = MEM(r8+r4)
      f5 = MEM(r2+r4)
      f6 = f3 * f5
      f11 = f11 + f6
      r4 = r4 + 4
      f3 = MEM(r8+r4)
      f5 = MEM(r2+r4)
      f6 = f3 * f5
      f12 = f12 + f6
      r4 = r4 + 4
      f3 = MEM(r8+r4)
      f5 = MEM(r2+r4)
      f6 = f3 * f5
      f13 = f13 + f6
      r4 = r4 + 4
      ble (r4 r3)  L1
      f1 = f11 + f12
      f1 = f1 + f13
```

After Induction Optimization

```
      r80 = r8
      r20 = r2
L1:   f3 = MEM(r80+0)
      f5 = MEM(r20+0)
      f6 = f3 * f5
      f11 = f11 + f6
      f3 = MEM(r80+4)
      f5 = MEM(r20+4)
      f6 = f3 * f5
      f12 = f12 + f6
      f3 = MEM(r80+8)
      f5 = MEM(r20+8)
      f6 = f3 * f5
      f13 = f13 + f6
      r80 = r80 + 12
      r20 = r20 + 12
      ble (r80 r31)  L1
```

After Renaming

```
L1:   f31 = MEM(r80+0)
      f51 = MEM(r20+0)
      f61 = f31 * f51
      f11 = f11 + f61
      f32 = MEM(r80+4)
      f52 = MEM(r20+4)
      f62 = f32 * f52
      f12 = f12 + f62
      f33 = MEM(r80+8)
      f53 = MEM(r20+8)
      f63 = f33 * f53
      f13 = f13 + f63
      r80 = r80 + 12
      r20 = r20 + 12
      ble (r80 r31)  L1
```

ResMII_eff: 1 to 0.66
RecMII_eff: 2 to 0.66

IMPACT

Related Work

- Unrolling-based optimization for acyclic scheduling
  - Multiflow optimizer [Lowney et al. 93]
  - Predictive Commoning [O'Brien et al. 90]
  - IMPACT ILP optimizer [Mahlke et al. 92]
- Optimization of software-pipelined loops
  - Cydrome Cydra-5 loop optimizer [Dehnert et al. 89]
  - Blocked Back-Substitution [Schlansker and Kathail 93]
  - Control Height Reduction [Schlansker et al. 94]
  - Expanded Virtual Registers [Rau 92]

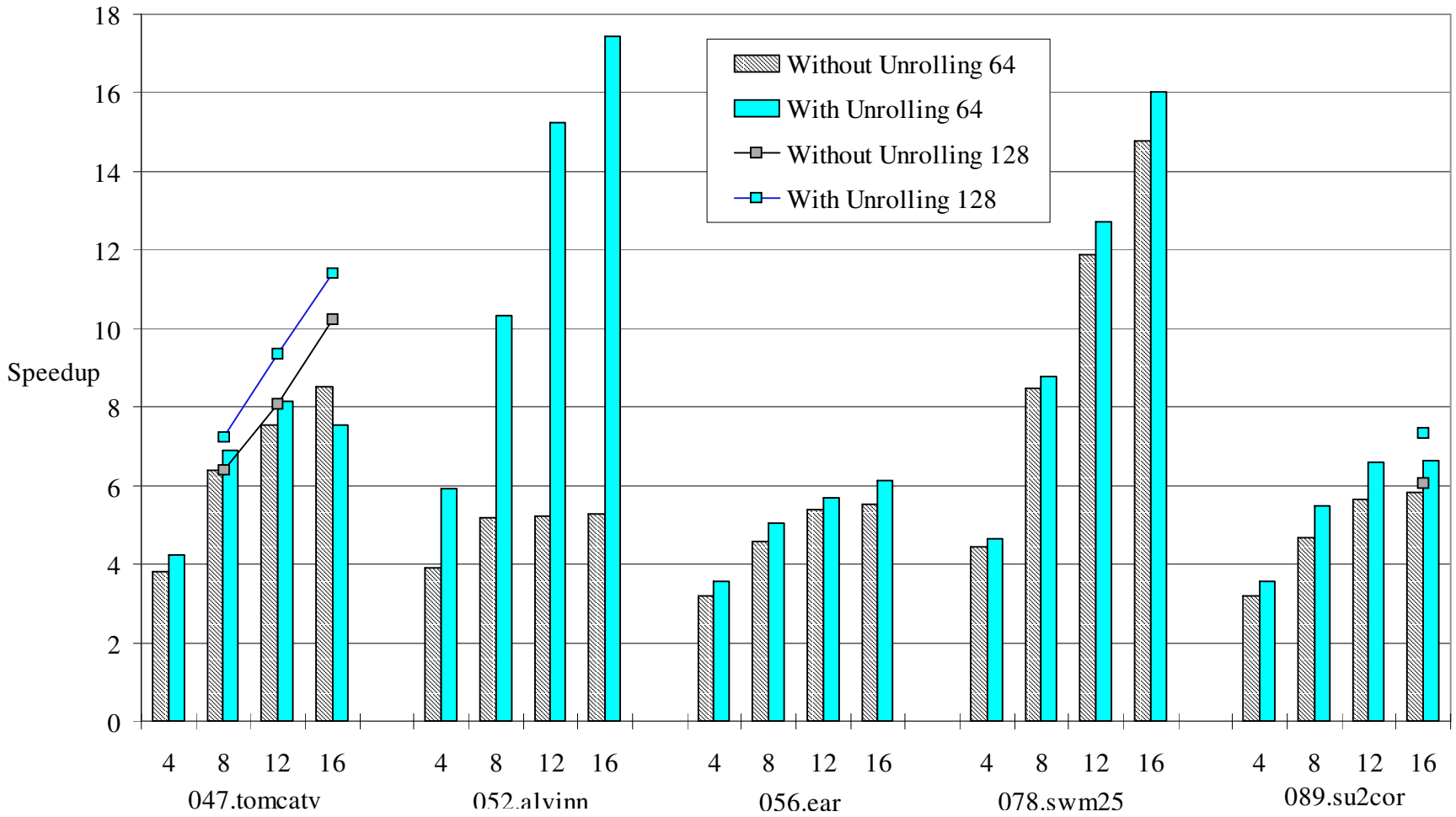
Experiments

- IMPACT compiler
  - data dependence analysis
    - Omega test [Pugh 92]
    - interprocedural alias analysis
  - modulo scheduler
    - single-basic-block DO and WHILE loops
    - single-node recurrences
    - no function calls
    - Modulo Variable Expansion
- Five SPEC CFP92 benchmarks
  - alvinn, swm256: 99%
  - tomcatv, su2cor, ear: 90%, 80%, 75%

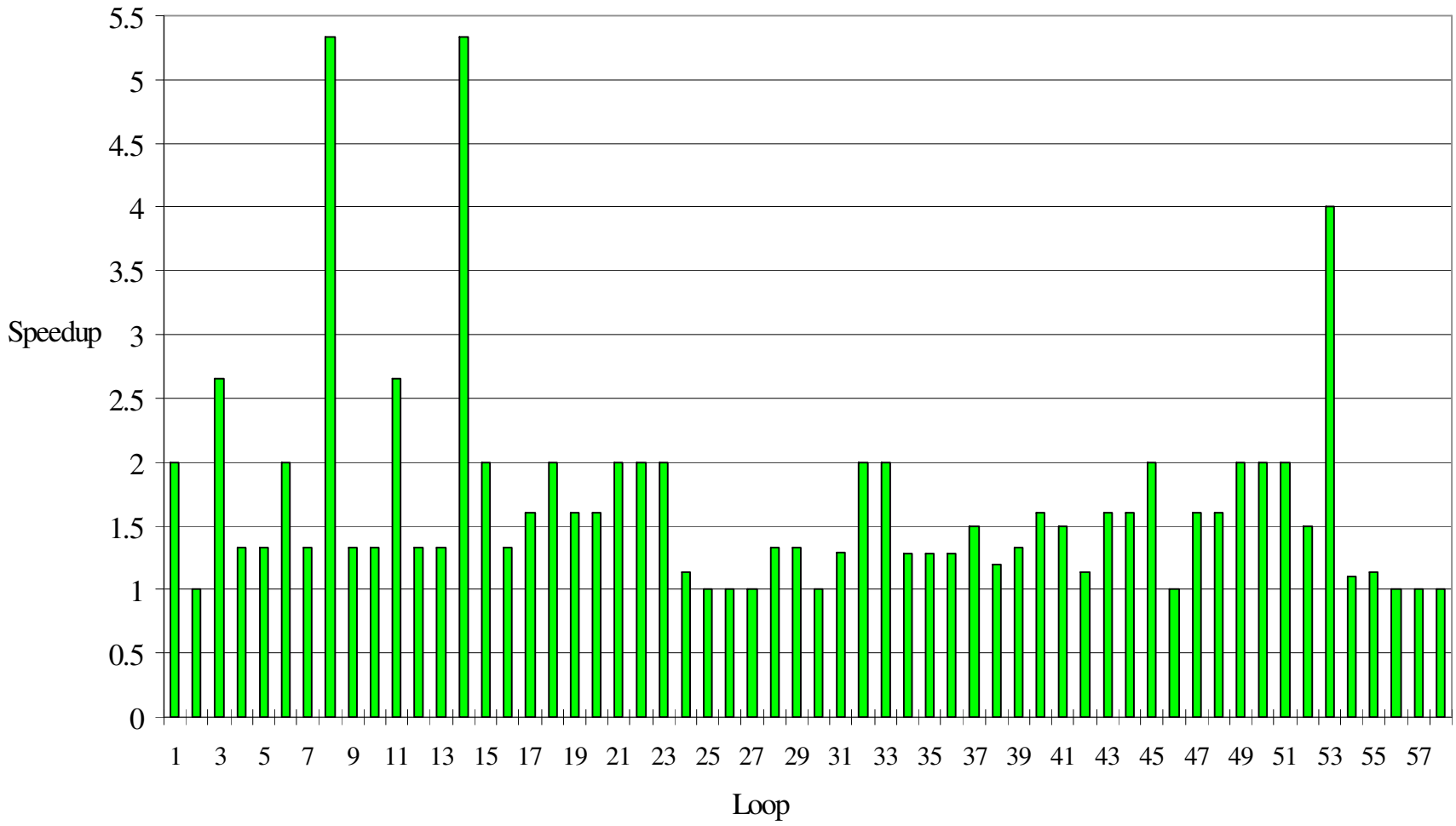
Experiments

- Processor model
  - 4- to 16-issue
  - no restrictions on the grouping of instructions
  - HP PA-7100 instruction latencies
  - 64 integer and 64 floating-point registers
    - 128 floating-point registers where noted
- Emulation to verify code correctness
- Profiled to find execution count for each basic block
- Execution time calculated from scheduler cycle count for each block

Benchmark Performance



Loop Performance



Loop Statistics

Measurement	Statistic	Original	Unrolled
Static Instructions	Minimum	5	14
	Maximum	76	137
	Mean	22	65
Unroll Amount	Minimum	NA	1
	Maximum	NA	8
	Mean	NA	4.7
	Median	NA	4
Effective MII	Minimum	1	0.25
	Maximum	5	5
	Mean	1.9	1.3
Speedup	Minimum	NA	1
	Maximum	NA	5.33
	Mean	NA	1.7

Conclusion

- Summary
  - unrolling prior to modulo scheduling
    - allows a non-integral effective II
    - removes the one-iteration-per-cycle limit on throughput
    - provides additional opportunities for optimization
    - produces speedups of 4% to more than 200%
- Future Work
  - balance optimization with register pressure
    - introduce a register-constrained lower bound on the MII
  - determine the optimal number of times to unroll
  - measure code expansion and effect on I-cache