
Modulo Scheduling

A Class of Methods for Software Pipelining Loops

Software Pipelining?

- A technique for scheduling instructions to exploit ILP in inner loops
 - many programs spend most of their time there
 - parallelism between loop iterations
 - e.g. DOALL loops

Case Study: Example Code

Source Code

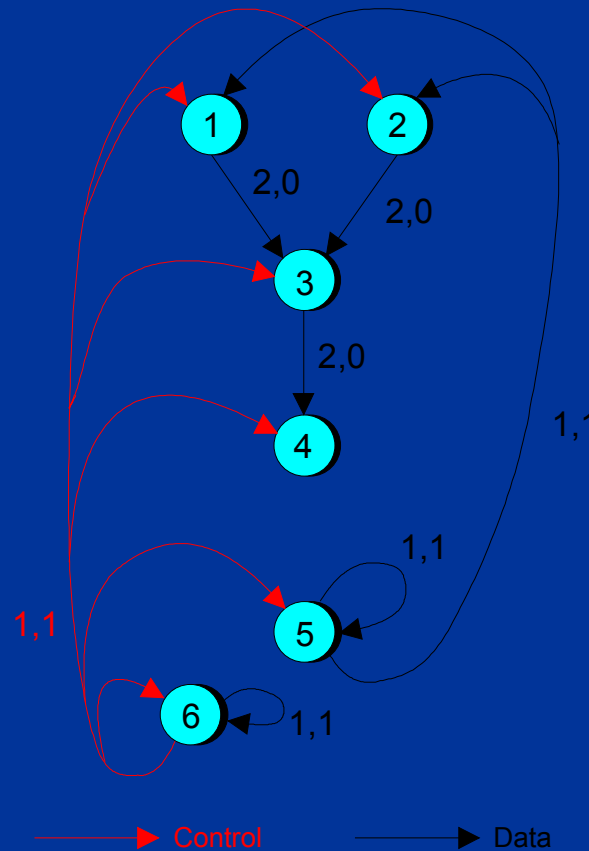
```
initialize C to 0.0
for (j=0; j<m; j++) {
    for (i=0; i<n; i++) {
        C[i] = C[i] + A[j] * B[j][i]
    }
}
```

Assembly Code for Inner Loop

Inst.	Assembly	Register Contents
1	L1: f1 = MEM(r8+r4)	f1 = C[i]
2	f5 = MEM(r2+r4)	f3 = A[j]
3	f6 = f3 * f5	f5 = B[j][i]
4	f1 = f1 + f6	r8 = &C[0]
5	r4 = r4 + 4	r2 = &B[j][0]
6	bgt ((--r5) 0) L1	r4 = 4*i
		r5 = n

Case Study: Dependence

Dependence Graph for Inner Loop



Case Study: Scheduling

- Processor model
 - 6-issue
 - 3 load/store units
 - 2 floating-point units
 - 1 branch per cycle
- Schedule for a single iteration
 - 5 cycles per iteration

Loop Unrolling

Original Assembly Code

Inst.	Assembly
1	L1: f1 = MEM(r8+r4)
2	f5 = MEM(r2+r4)
3	f6 = f3 * f5
4	f1 = f1 + f6
5	r4 = r4 + 4
6	bgt ((--r5) 0) L1

Unrolled Assembly Code

Inst.	Assembly
1 ₁	L1: f11 = MEM(r8+r41)
2 ₁	f51 = MEM(r2+r41)
3 ₁	f61 = f3 * f51
4 ₁	f11 = f11 + f61
5 ₁	r42 = r41 + 4
6 ₁	ble ((--r5) 0) exit
1 ₂	f12 = MEM(r8+r42)
2 ₂	f52 = MEM(r2+r42)
3 ₂	f62 = f3 * f52
4 ₂	f12 = f12 + f62
5 ₂	r41 = r42 + 4
6 ₂	bgt ((--r5) 0) L1

Acyclic Scheduling

Two Copies

0	1 ₁	2 ₁	5 ₁			
1		1 ₂	2 ₂	5 ₂		
2		3 ₁				
3			3 ₂			
4		4 ₁	6 ₁			
5			4 ₂	6 ₂		

3 cycles per iteration

Eight Copies

0	1 ₁	2 ₁	5 ₁			
1		1 ₂	2 ₂	5 ₂		
2		3 ₁	1 ₃	2 ₃	5 ₃	
3			3 ₂	1 ₄	2 ₄	5 ₄
4	5 ₅	4 ₁	6 ₁	3 ₃	1 ₅	2 ₅
5	2 ₆	5 ₆	4 ₂	6 ₂	3 ₄	1 ₆
6	1 ₇	2 ₇	5 ₇	4 ₃	6 ₃	3 ₅
7	3 ₆	1 ₈	2 ₈	5 ₈	4 ₄	6 ₄
8	6 ₅	3 ₇				4 ₅
9	4 ₆	6 ₆	3 ₈			
10		4 ₇	6 ₇			
11			4 ₈	6 ₈		

1.5 cycles per iteration

Outline

- Basic modulo scheduling concepts
- Issues in generating modulo scheduled code
- Advanced modulo scheduling topics

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)

Initiation Interval

- 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 Π

Resource MII

- Enforced using a Modulo Resource Table (MRT)
 - II rows and one column per resource
 - schedule a single iteration

Modulo Resource Table

- The MRT must contain a sufficient amount of each resource
 - Resource-constrained lower bound on Π (ResMII)

Resource-constrained MII

Assembly Code Loop

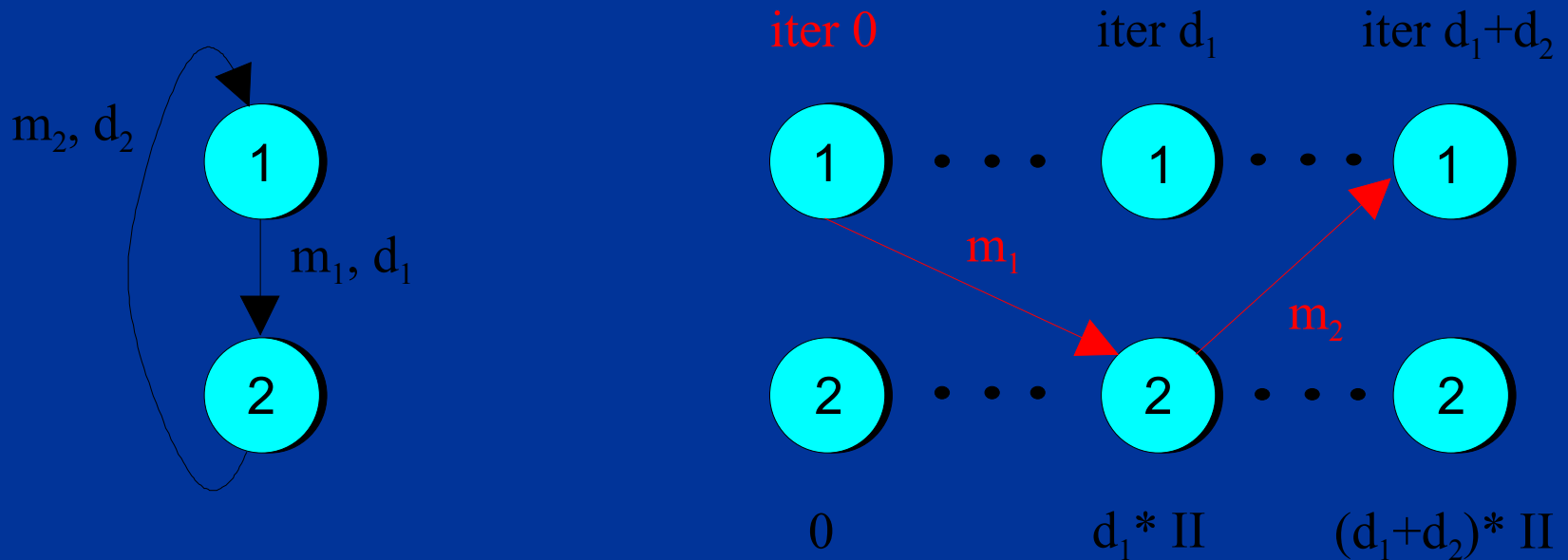
```
L1: f3 = MEM(r8+r4)
    f5 = MEM(r2+r4)
    f6 = f3 * f5
    MEM(r9+r4) = f6
    r4 = r4 +4
    ble (r4 r7) L1
```

MRT

	Slot 1	Slot 2
0		
1		
2		

ResMII = 3

Recurrence-constrained MII



$$\text{Delay}(c) = m_1 + m_2$$

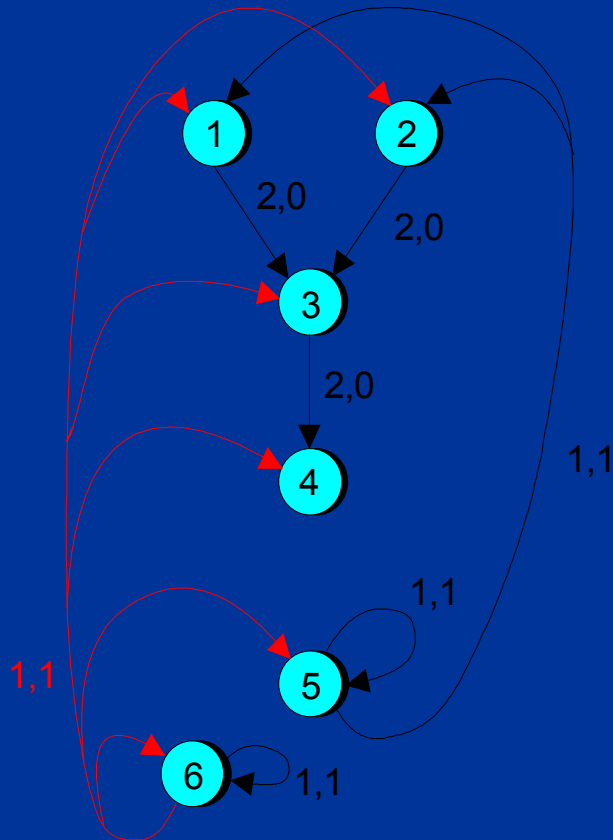
$$\text{Distance}(c) = d_1 + d_2$$

Iteration Start Time

$$\text{Delay}(c) \leq \text{Distance}(c) * II$$

Scheduling: Example Loop

Dependence Graph for Inner Loop



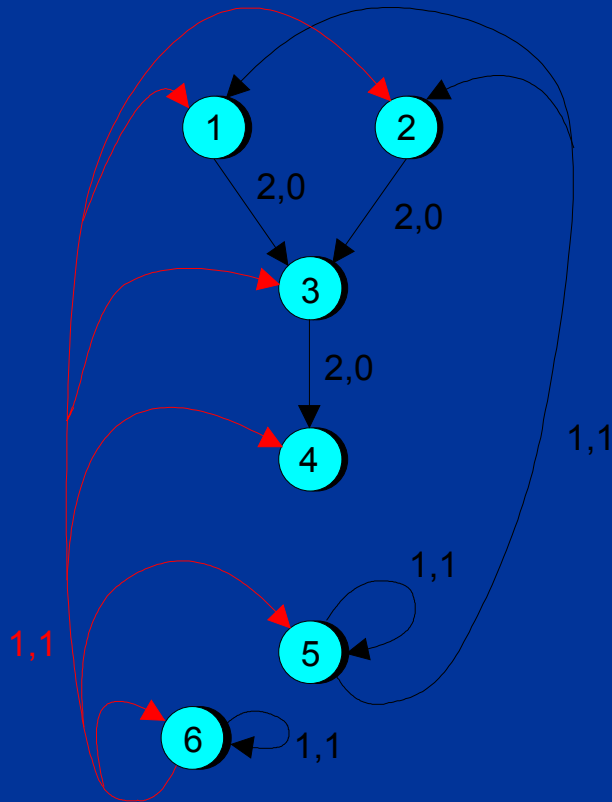
- $\text{ResMII} = \text{RecMII} = 1$

MRT

slot 0	slot 1	slot 2	slot 3	slot 4	slot 5
instr. 1	instr. 2			instr. 5	instr. 6
issue 0	issue 0			issue 0	issue 0

Scheduling the Example

Dependence Graph for Inner Loop



- Wrap around

slot 0	slot 1	slot 2	slot 3	slot 4	slot 5
instr. 1	instr. 2	instr. 3		instr. 5	instr. 6
issue 0	issue 0	issue 2		issue 0	issue 0

Scheduling: Example Loop

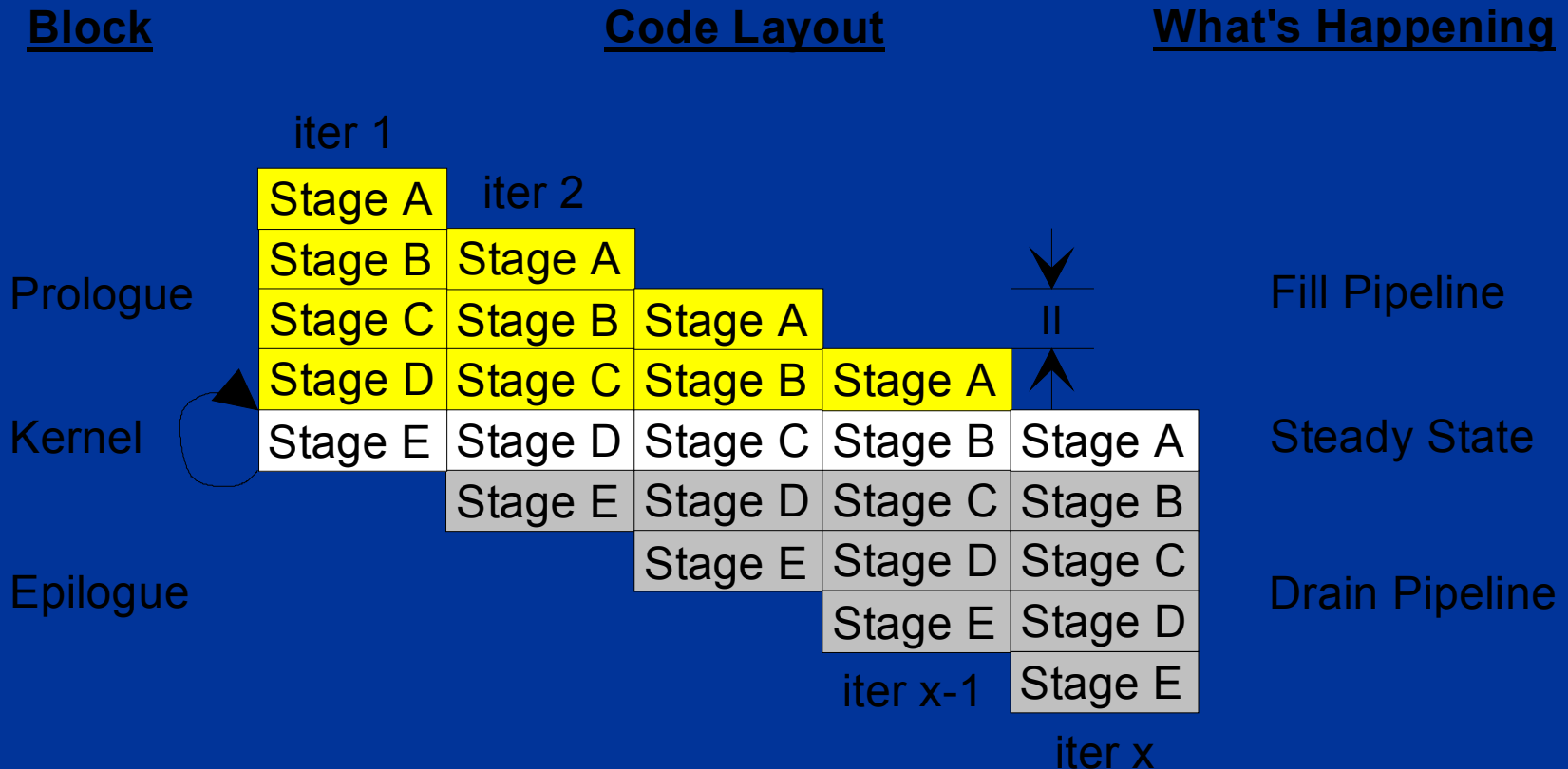
Schedule for a Single Iteration

Stage	slot 0	slot 1	slot 2	slot 3	slot 4	slot 5
A	instr. 1 issue 0	instr. 2 issue 0			instr. 5 issue 0	instr. 6 issue 0
B						
C			instr. 3 issue 2			
D						
E				instr. 4 issue 4		

MRT

slot 0	slot 1	slot 2	slot 3	slot 4	slot 5
instr. 1 issue 0	instr. 2 issue 0	instr. 3 issue 2	instr. 4 issue 4	instr. 5 issue 0	instr. 6 issue 0

Code Generation



Outline

- Basic modulo scheduling concepts
- Issues in generating modulo scheduled code
- Advanced modulo scheduling topics

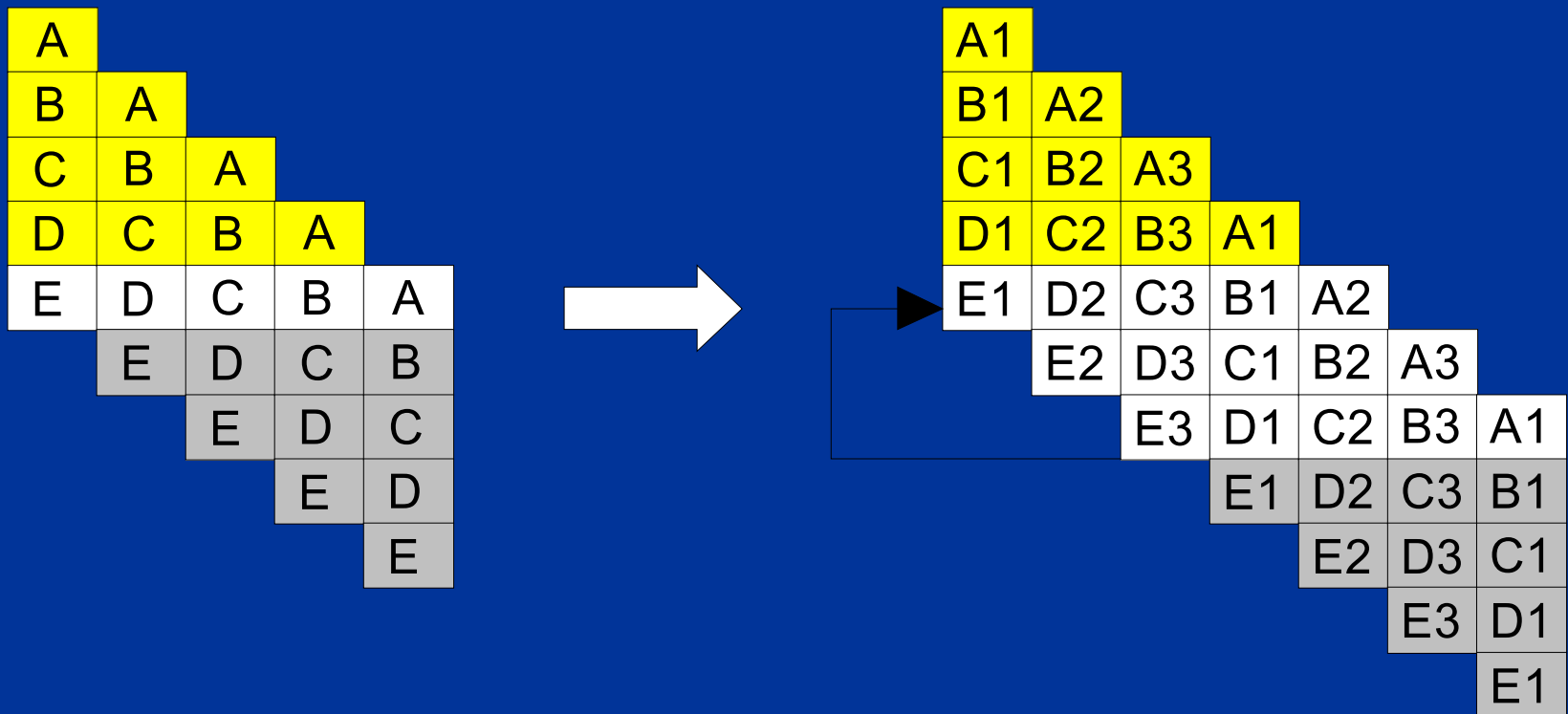
Issue #1 - Overlapped Lifetimes

Stage	slot 0	slot 1	slot 2	slot 3	slot 4	slot 5
A	instr. 1 issue 0	instr. 2 issue 0			instr. 5 issue 0	instr. 6 issue 0
B	instr. 1 issue 1					
C	instr. 1 issue 2		instr. 3 issue 2			
D						
E				instr. 4 issue 4		

Overlapping Liveranges

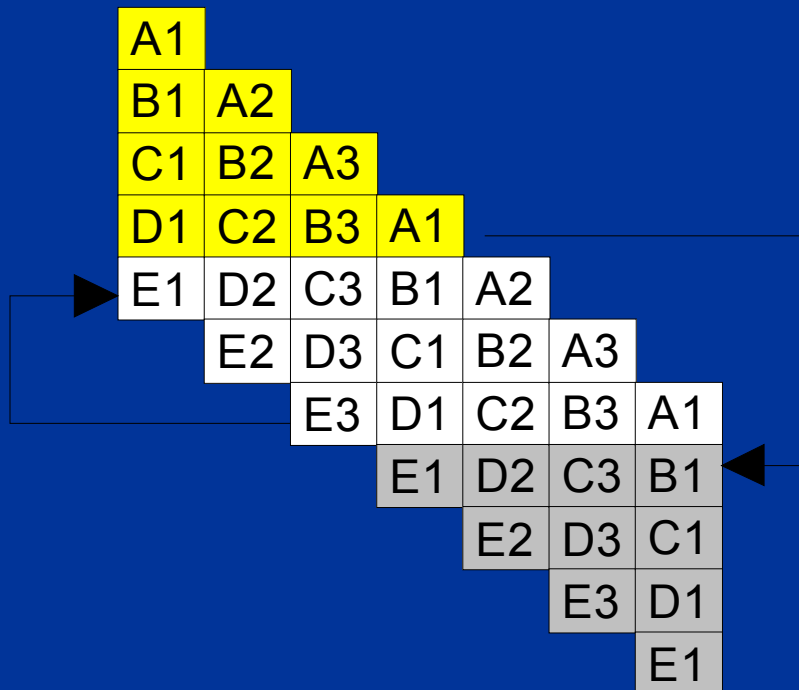
- Modulo Variable Expansion
 - $k = \text{ceiling}(v/l)$
 - in example, $k = 3$
 - k is degree of unrolling
 - v is length of longest lifetime
 - account for ordering in slots
- Rotating Registers

Kernel Unrolling for MVE



Issue #2 - Correct Code for All Possible Trip Counts

- Assume the only exits from the pipeline are at the end of the prologue and the end of the kernel as shown

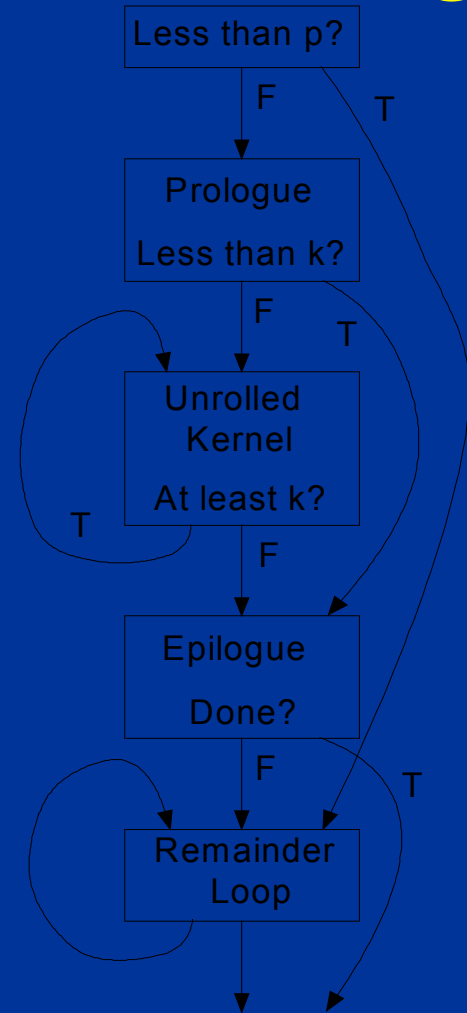
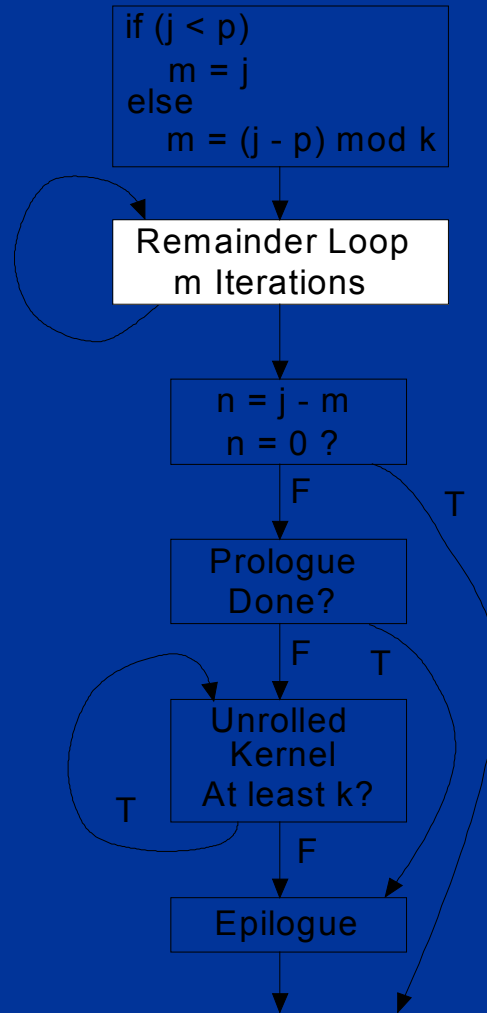


All Possible Trip Counts

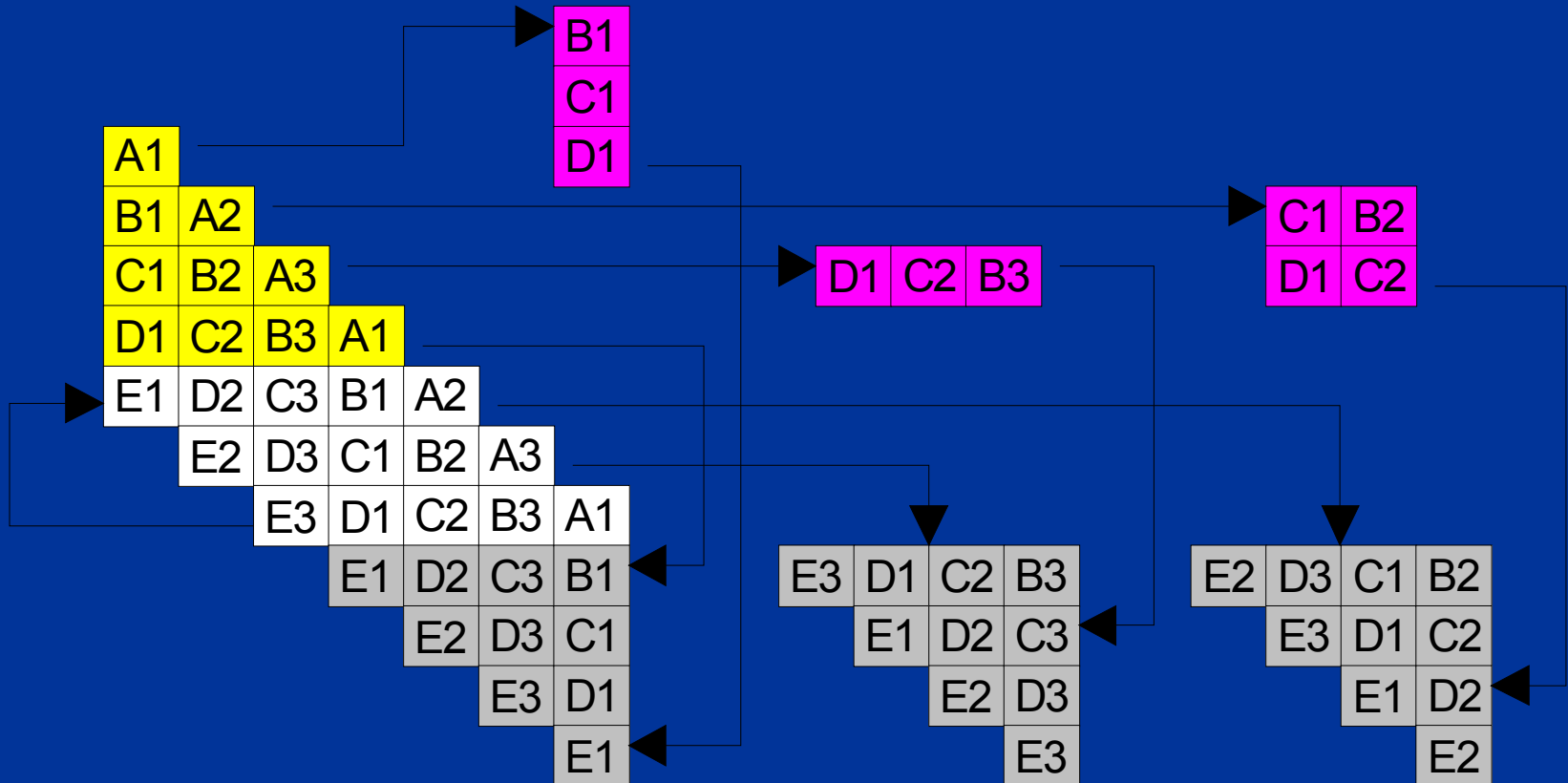
- Pipeline can only execute certain numbers of iterations, $n = k*i + p$
- k is the degree of kernel unroll
- i is an integer greater than or equal to 0
- p is the number of iterations started in the prologue

Pre- and Post-Conditioning

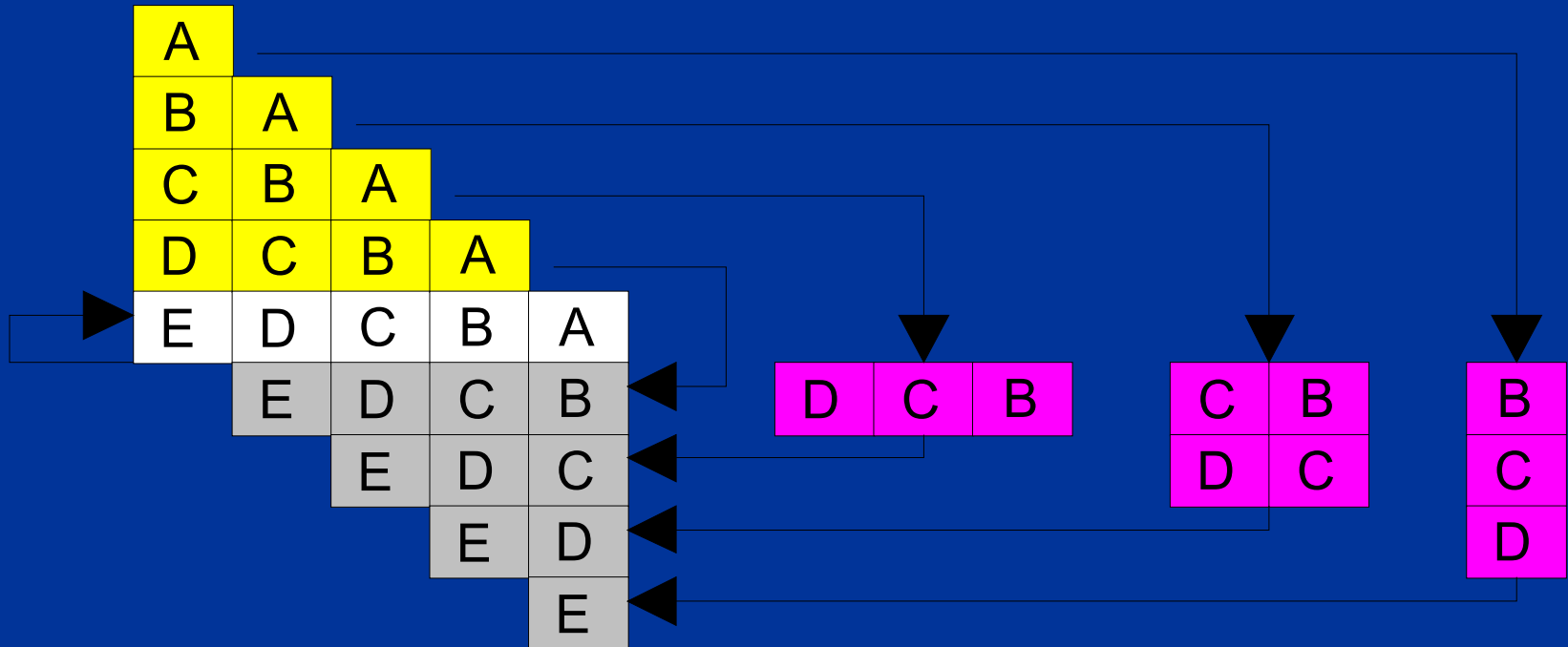
j = original #
of iterations



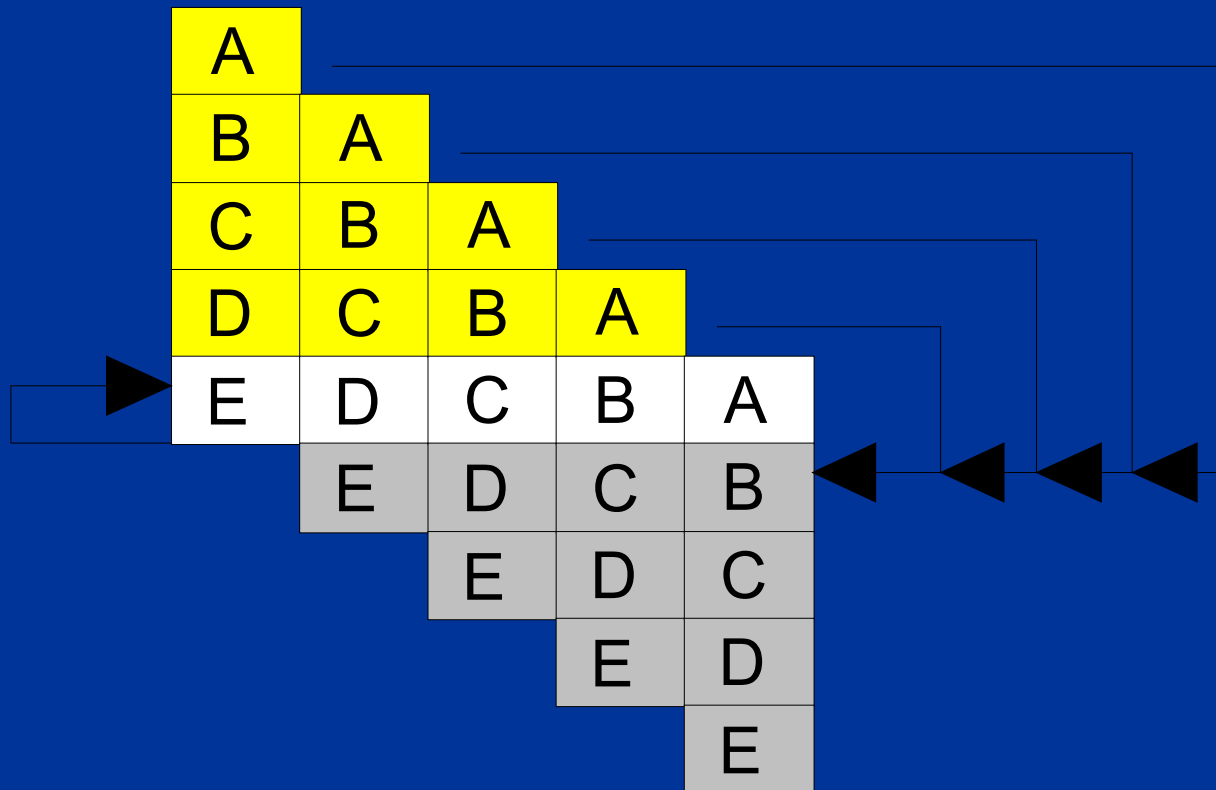
Multiple Epilogues



With Rotating Registers



With Rotating Registers and Predicated Execution



Kernel Only Code



Issue #3 - Iterative Modulo Scheduling

- The MII may not be an achievable lower bound
 - instructions along a recurrence delayed by resource conflicts
 - complex patterns of resource usage

Iterative Modulo Scheduling

- Increment M till reaching a schedule
- Even if a schedule exists at M , a list scheduler may fail to find it
 - allow limited unscheduling and rescheduling
 - limit the number of scheduling attempts for each instruction

Outline

- Basic modulo scheduling concepts
- Issues in generating modulo scheduled code
- Advanced modulo scheduling topics

Unrolling-Based Optimization

- Unroll loop before modulo scheduling
- Reduce resource usage and dependence height

Unrolling-based Optimization

- Motivation

- It is restricted to be an integer and to be no smaller than one

- performance degradation due to rounding

- throughput limited to one iteration per cycle

Unrolling-Based Optimization

- Motivation (cont.)
 - Some optimizations not possible without unrolling
 - exit branch and induction op removal, blocked back-substitution
 - accumulator expansion, cyclic copy propagation without EVRs

Unrolling-Based Optimizations

- Performance: Micro-28 paper
 - improvements range from 4% to more than 200%
 - 4- to 16-issue processors
 - tomcatv, alvinn, ear, swm256, su2cor

