Introduction to Software Pipelining

- Basic Idea of Software Pipelining

A Motivating Example

- Machine model: one memory access (1-cycle), one arithmetic operation (2-cycle) in parallel
- Source code: Do-All style loop

```
for (i=0; i < n; i++)
    A[i] = A[i] * b + c
```

- Code for one iteration: 6 cycles/iteration

cycle 1: `Read`
cycle 2: `Multiply`
cycle 3:
cycle 4: `Add`
cycle 5:
cycle 6: `Write`

Loop unrolling

- Unrolling replaces the body of the loop by several copies of the body and adjusts loop-control code
  - Degree of unrolling = number of loop bodies

- Unrolling once and schedule: 7 cycles/2 iterations

```
1: Read
2: Mul      Read
3:          Mul
4: Add
5:          Add
6: Write
7:          Write
```

- Unrolling twice and schedule: 10 cycles/3 iterations

Impact of Unrolling

- What would be the optimal performance of this loop?
  - 2 cycles/iteration (why? Consider resource constraints only)

- Impact of unrolling: Let u be the degree of unrolling
  - Execution Time of unrolled loop = $6 + 2(u - 1) = 4 + 2u$
  - Optimal execution time = $2u$
  - Efficiency = $\frac{2u}{4+2u}$
  - Efficiency = 90 % $\Rightarrow$ $u = 18$

- More you unroll, it become better, but the code size increases substantially

Software Pipelining

An optimization technique that can schedule instructions beyond loop iteration boundaries

- By overlapping iterations in a pipelined fashion
- Multiple iterations can be executed in parallel
- Future iterations can initiate before current ones finish

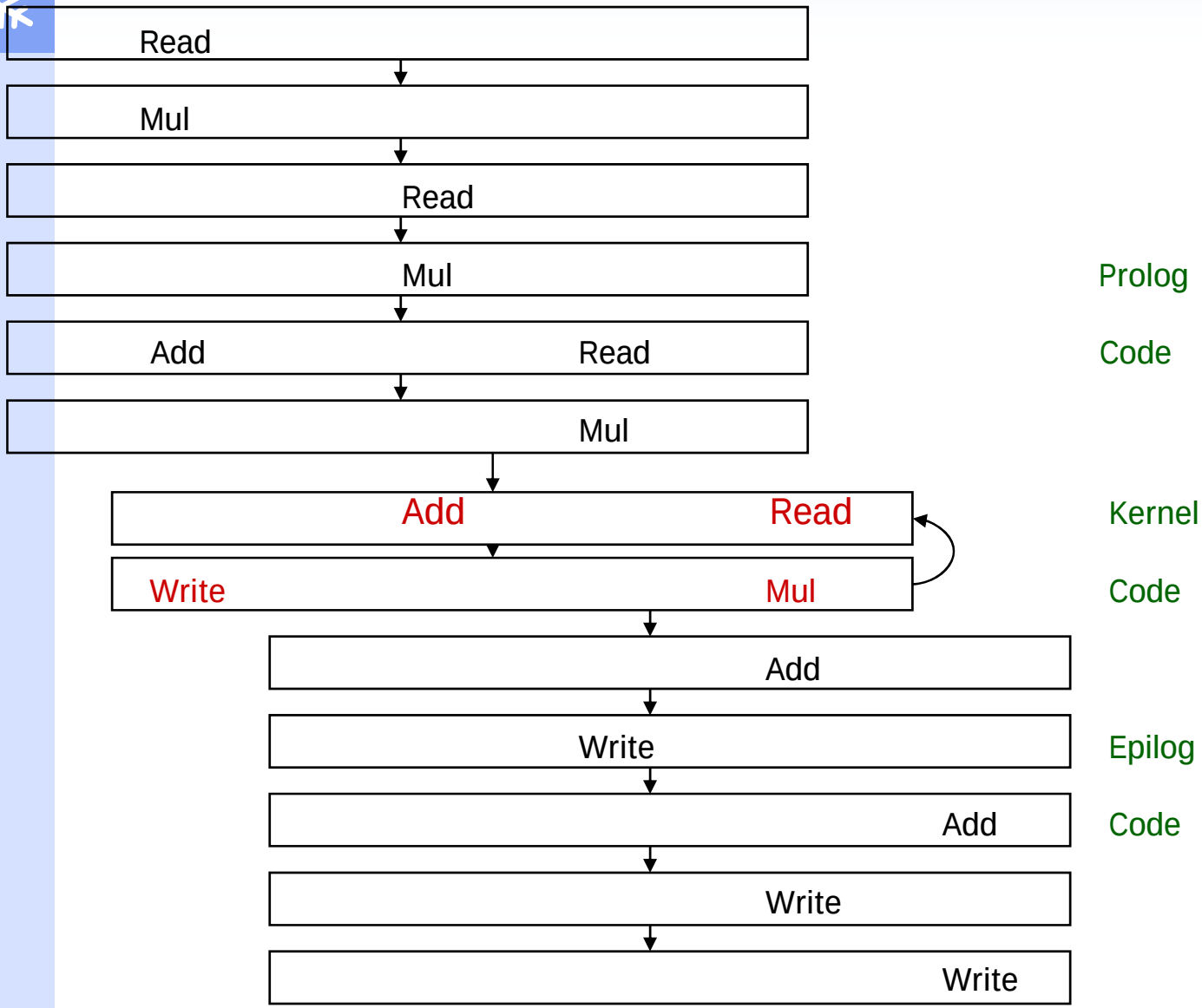
Generating a pipelined schedule in overlapped iterations

- Must find a pattern of code composed of multiple iterations that can be executed repeatedly, which is called a kernel

Finding a Kernel in Overlapped Code

	1st iteration	2nd iteration	3rd iteration	4th iteration	5th iteration	
1:	Read					
2:	Mul					
3:		Read				
4:		Mul				
5:	Add		Read			
6:			Mul			
7:		Add		Read		: repeated
8:	Write			Mul		: pattern
9:			**Add**		**Read**	| repeated
10:		**Write**			**Mul**	| pattern
11:				Add		
12:			Write			
13:					Add	
14:				Write		
15:						
16:					Write	

Generating a Pipelined Schedule



Software Pipelined Loop and II

- A software pipelined loop is composed of:
  - Prolog: pipeline startup code
  - Kernel: repeated pattern that is executed repetitively
  - Epilog: pipeline drain code

- Initiation interval (II)
  - Interval with which the next iteration start after the current iteration initiates
  - Equals to the cycle length of the kernel
  - In our example schedule, II = 2 cycles

Benefit of Software Pipelining

Unlike unrolling, software pipelining can give you an optimal result

Code size is much smaller than unrolling

Schedule of each iteration

- Schedule of each iteration is identical
  - For finding a pattern easily and quickly
- Locally compacted code might not be globally optimal

Software Pipelining of Do-Across Loops

- Source Code

```
for (i=0; i < n; i++)
      Sum = Sum + A[i]
      A[i] = A[i] * b
```

```
1: read
2: Mult
3: Add
4: Write
```

- Software pipelined Code

```
1:   Read
2:   Mul
3:   Add      Read     | kernel
4:   Write    Mul      |
5:            Add
6:            Write
```

  - Recurrences can also be parallelized
  - Harder than Do-All loops

Software Pipelining Techniques

There are many S/W pipelining techniques out there, but there are only two practical techniques

- Modulo scheduling (MS)
- Enhanced pipeline scheduling (EPS)

MS is based on code placement, while EPS is based on code motion