Instruction Scheduling (Part 2)

Software Pipelining

Patrick Carribault
patrick@ices.utexas.edu

CS 380C: Advanced Compiler Techniques

Tuesday, October 23th 2007

Lecture Overview

Code Generator

- Back end part of compiler (code generator)
- Instruction scheduling
- Register allocation

Instruction Scheduling

- Input: set of instructions
- Output: total order on that set

Lecture Outline

Previous Lecture

1. Introduction to instruction scheduling
2. Representation of data dependences and resource constraints
3. Acyclic scheduling: list scheduling

Today

- Loop scheduling: definition of software pipelining
- Parameters of software-pipelined schedules
- Heuristics: *modulo scheduling*
- Hardware support for code generation of software-pipelined loops

Introduction to Software Pipelining

Loop Scheduling

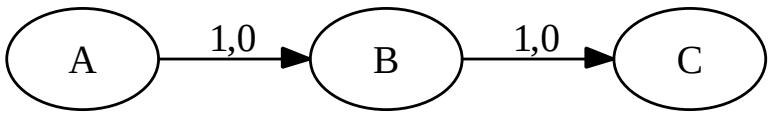
- Apply list scheduling on the loop body
- Ignore dependence distance > 0
- No iteration overlapping $\Rightarrow$ any schedule respects the dependence distances
- **But**: exploit only intra-iteration parallelism

How to benefit from inter-iteration parallelism?

1. Unroll the loop before scheduling (or while scheduling)
2. Overlap consecutive iterations in a continuous flow
   $\Rightarrow$ Software Pipelining

Example 1

DDG and reservation tables:



A	r0	r1	r2
0	X		

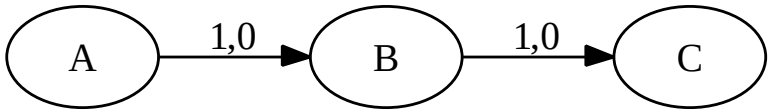
B	r0	r1	r2
0		X	

C	r0	r1	r2
0			X

- A valid simple schedule?

Example 1

DDG and reservation tables:



A	r0	r1	r2
0	X		

B	r0	r1	r2
0		X	

C	r0	r1	r2
0			X

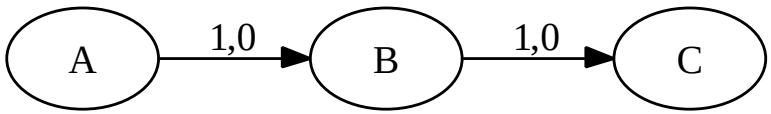
- A valid simple schedule? $\Rightarrow$ Use list scheduling on loop body

Cycle	Schedule

Cycle	r0	r1	r2

Example 1

DDG and reservation tables:



A	r0	r1	r2
0	X		

B	r0	r1	r2
0		X	

C	r0	r1	r2
0			X

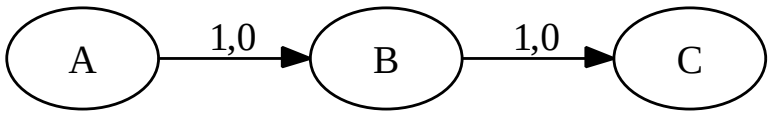
- A valid simple schedule? $\Rightarrow$ Use list scheduling on loop body

Cycle	Schedule
0	A

Cycle	r0	r1	r2
0	X		

Example 1

DDG and reservation tables:



A	r0	r1	r2
0	X		

B	r0	r1	r2
0		X	

C	r0	r1	r2
0			X

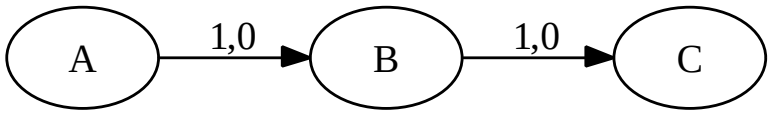
- A valid simple schedule? $\Rightarrow$ Use list scheduling on loop body

Cycle	Schedule
0	A
1	B

Cycle	r0	r1	r2
0	X		
1		X	

Example 1

DDG and reservation tables:



A	r0	r1	r2
0	X		

B	r0	r1	r2
0		X	

C	r0	r1	r2
0			X

- A valid simple schedule? $\Rightarrow$ Use list scheduling on loop body

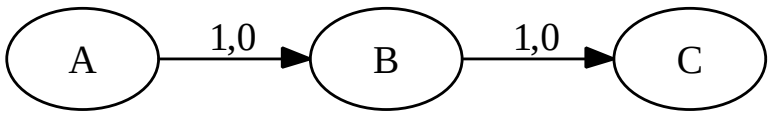
Cycle	Schedule
0	A
1	B
2	C

Cycle	r0	r1	r2
0	X		
1		X	
2			X

- Length of 3, but inter-iteration parallelism available

Example 1 – cont.

DDG and reservation tables:



A	r0	r1	r2
0	X		

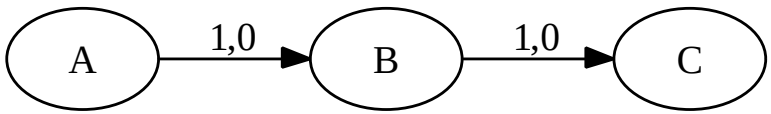
B	r0	r1	r2
0		X	

C	r0	r1	r2
0			X

- Schedule with overlapped iterations?

Example 1 – cont.

DDG and reservation tables:



A	r0	r1	r2
0	X		

B	r0	r1	r2
0		X	

C	r0	r1	r2
0			X

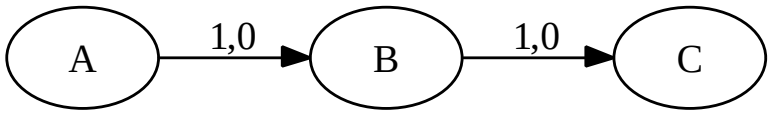
- Schedule with overlapped iterations?

Cycle	Schedule			
	0	1	2	3

Cycle	r0	r1	r2

Example 1 – cont.

DDG and reservation tables:



A	r0	r1	r2
0	X		

B	r0	r1	r2
0		X	

C	r0	r1	r2
0			X

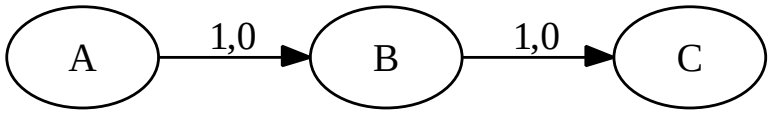
- Schedule with overlapped iterations?

Cycle	Schedule			
	0	1	2	3
0	A			

Cycle	r0	r1	r2
0	X		

Example 1 – cont.

DDG and reservation tables:



A	r0	r1	r2
0	X		

B	r0	r1	r2
0		X	

C	r0	r1	r2
0			X

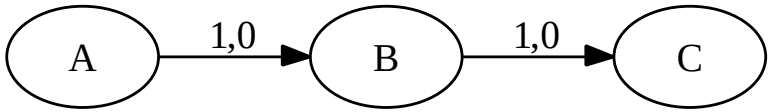
- Schedule with overlapped iterations?

Cycle	Schedule			
	0	1	2	3
0	A			
1	B	A		

Cycle	r0	r1	r2
0	X		
1	X	X	

Example 1 – cont.

DDG and reservation tables:



A	r0	r1	r2
0	X		

B	r0	r1	r2
0		X	

C	r0	r1	r2
0			X

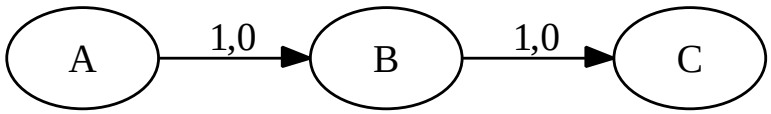
- Schedule with overlapped iterations?

Cycle	Schedule			
	0	1	2	3
0	A			
1	B	A		
2	C	B	A	

Cycle	r0	r1	r2
0	X		
1	X	X	
2	X	X	X

Example 1 – cont.

DDG and reservation tables:



A	r0	r1	r2
0	X		

B	r0	r1	r2
0		X	

C	r0	r1	r2
0			X

- Schedule with overlapped iterations?

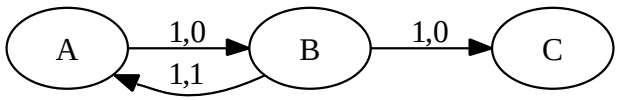
Cycle	Schedule			
	0	1	2	3
0	A			
1	B	A		
2	C	B	A	
3		C	B	A

Cycle	r0	r1	r2
0	X		
1	X	X	
2	X	X	X
3	X	X	X

- Kernel (1 cycle), depth of 3

Example 2

DDG and reservation tables:



A	r0	r1	r2
0	X		

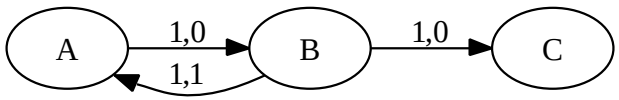
B	r0	r1	r2
0		X	

C	r0	r1	r2
0			X

- Schedule with overlapped iterations?

Example 2

DDG and reservation tables:



A	r0	r1	r2
0	X		

B	r0	r1	r2
0		X	

C	r0	r1	r2
0			X

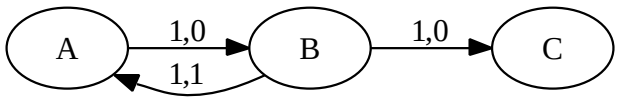
- Schedule with overlapped iterations?

Cycle	Schedule			
	0	1	2	3

Cycle	r0	r1	r2

Example 2

DDG and reservation tables:



A	r0	r1	r2
0	X		

B	r0	r1	r2
0		X	

C	r0	r1	r2
0			X

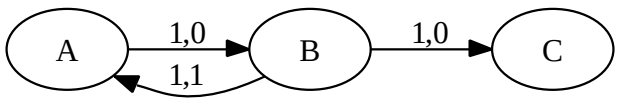
- Schedule with overlapped iterations?

Cycle	Schedule			
	0	1	2	3
0	A			

Cycle	r0	r1	r2
0	X		

Example 2

DDG and reservation tables:



A	r0	r1	r2
0	X		

B	r0	r1	r2
0		X	

C	r0	r1	r2
0			X

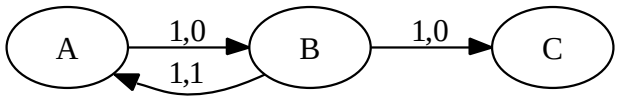
- Schedule with overlapped iterations?

Cycle	Schedule			
	0	1	2	3
0	A			
1	B			

Cycle	r0	r1	r2
0	X		
1		X	

Example 2

DDG and reservation tables:



A	r0	r1	r2
0	X		

B	r0	r1	r2
0		X	

C	r0	r1	r2
0			X

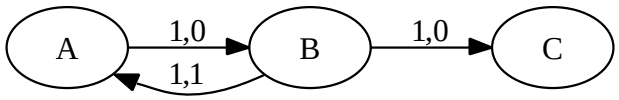
- Schedule with overlapped iterations?

Cycle	Schedule			
	0	1	2	3
0	A			
1	B			
2	C	A		

Cycle	r0	r1	r2
0	X		
1		X	
2	X		X

Example 2

DDG and reservation tables:



A	r0	r1	r2
0	X		

B	r0	r1	r2
0		X	

C	r0	r1	r2
0			X

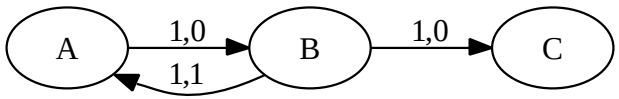
- Schedule with overlapped iterations?

Cycle	Schedule			
	0	1	2	3
0	A			
1	B			
2	C	A		
3		B		

Cycle	r0	r1	r2
0	X		
1		X	
2	X		X
3		X	

Example 2

DDG and reservation tables:



A	r0	r1	r2
0	X		

B	r0	r1	r2
0		X	

C	r0	r1	r2
0			X

- Schedule with overlapped iterations?

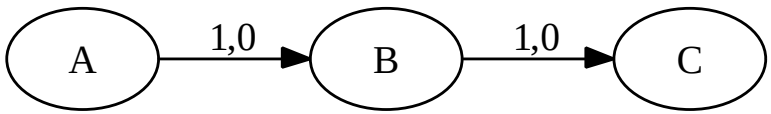
Cycle	Schedule			
	0	1	2	3
0	A			
1	B			
2	C	A		
3		B		
4		C	A	

Cycle	r0	r1	r2
0	X		
1		X	
2	X		X
3		X	
4	X		X

- Kernel (2 cycles), depth of 2

Example 3

DDG and reservation tables:



A	r0	r1
0	X	

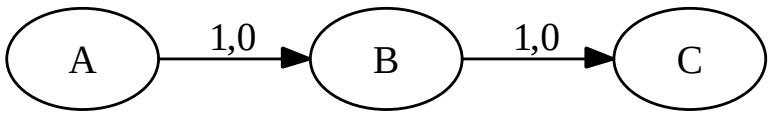
B	r0	r1
0	X	

C	r0	r1
0		X

- Schedule with overlapped iterations?

Example 3

DDG and reservation tables:



A	r0	r1
0	X	

B	r0	r1
0	X	

C	r0	r1
0		X

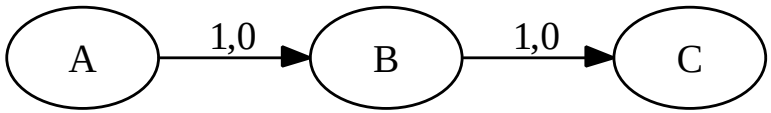
- Schedule with overlapped iterations?

Cycle	Schedule			
	0	1	2	3

Cycle	r0	r1

Example 3

DDG and reservation tables:



A	r0	r1
0	X	

B	r0	r1
0	X	

C	r0	r1
0		X

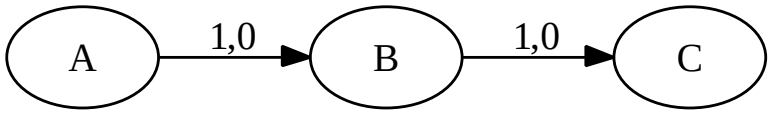
- Schedule with overlapped iterations?

Cycle	Schedule			
	0	1	2	3
0	A			

Cycle	r0	r1
0	X	

Example 3

DDG and reservation tables:



A	r0	r1
0	X	

B	r0	r1
0	X	

C	r0	r1
0		X

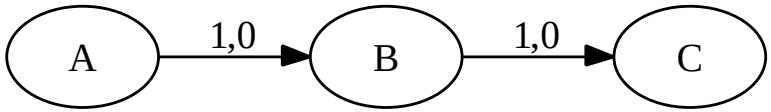
- Schedule with overlapped iterations?

Cycle	Schedule			
	0	1	2	3
0	A			
1	B			

Cycle	r0	r1
0	X	
1	X	

Example 3

DDG and reservation tables:



A	r0	r1
0	X	

B	r0	r1
0	X	

C	r0	r1
0		X

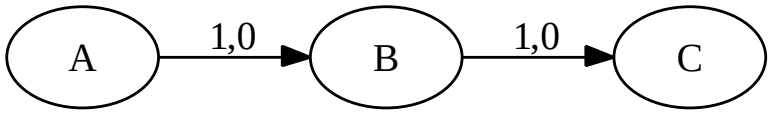
- Schedule with overlapped iterations?

Cycle	Schedule			
	0	1	2	3
0	A			
1	B			
2	C	A		

Cycle	r0	r1
0	X	
1	X	
2	X	X

Example 3

DDG and reservation tables:



A	r0	r1
0	X	

B	r0	r1
0	X	

C	r0	r1
0		X

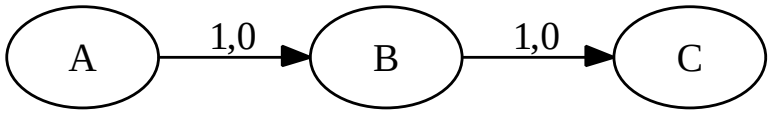
- Schedule with overlapped iterations?

Cycle	Schedule			
	0	1	2	3
0	A			
1	B			
2	C	A		
3		B		

Cycle	r0	r1
0	X	
1	X	
2	X	X
3	X	

Example 3

DDG and reservation tables:



A	r0	r1
0	X	

B	r0	r1
0	X	

C	r0	r1
0		X

- Schedule with overlapped iterations?

Cycle	Schedule			
	0	1	2	3
0	A			
1	B			
2	C	A		
3		B		
4		C	A	

Cycle	r0	r1
0	X	
1	X	
2	X	X
3	X	
4	X	X

- Kernel (2 cycles), depth of 2

Software Pipelining – Definition

Definition

- Do not wait for an iteration to finish to launch the next one
- Constant time between two consecutive iteration launches
  - Initiation Interval (or *II*)
- Need to respect constraints (including dependence distance)
- Prolog/Kernel/Epilog

Schedule

- Several iterations *alive* in the kernel (depth)
- One instruction scheduling $\Rightarrow$ one cycle and one iteration
  $\Rightarrow$ 2-dimensional schedule

Software Pipelining – Parameters

Performance

- Performance in cycles P with n iterations:

$$P = (n - 1) \times II + M$$

- Linear in $II \Rightarrow$ lower is better

Parameters

- Initiation Interval II
- Depth D
- Makespan M

Initiation Interval

Initiation Interval

- Time (in cycles) between 2 consecutive iteration launches
- Corresponds to the kernel size
- Shape the overall performance of pipelined schedule

Parameters influencing *II*

- Data dependences: *RecMII*:

$$RecMII = \max_{\forall circuits\ \theta} \left\lceil \frac{latency(\theta)}{distance(\theta)} \right\rceil$$

- Resource constraints: *ResMII*
- Minimum value *MII*:

$$MII = \max(ResMII, RecMII)$$

Data Dependences – RecMII

Dependence Constraint

- Let σ be the schedule date of instructions for one iteration, l is the latency and d the distance
- Consider a and b two instructions:
  - Intra-iteration constraint ($d(a, b) = 0$)

Data Dependences – RecMII

Dependence Constraint

- Let σ be the schedule date of instructions for one iteration, l is the latency and d the distance
- Consider a and b two instructions:
  - Intra-iteration constraint ($d(a, b) = 0$)

$$\sigma(a) + l(a, b) \leq \sigma(b)$$

  - Inter-iteration constaint (loop-carried dependences)

Data Dependences – RecMII

Dependence Constraint

- Let σ be the schedule date of instructions for one iteration, l is the latency and d the distance
- Consider a and b two instructions:
  - Intra-iteration constraint ($d(a, b) = 0$)

$$\sigma(a) + l(a, b) \leq \sigma(b)$$

  - Inter-iteration constaint (loop-carried dependences)

$$\sigma(a) + l(a, b) \leq \sigma(b) + II \times d(a, b)$$

MII due to dependence constraints on a circuit θ

Data Dependences – RecMII

Dependence Constraint

- Let σ be the schedule date of instructions for one iteration, l is the latency and d the distance
- Consider a and b two instructions:
  - Intra-iteration constraint ($d(a, b) = 0$)

$$\sigma(a) + l(a, b) \leq \sigma(b)$$

  - Inter-iteration constaint (loop-carried dependences)

$$\sigma(a) + l(a, b) \leq \sigma(b) + II \times d(a, b)$$

MII due to dependence constraints on a circuit θ

$$l(\theta) - II \times d(\theta) \leq 0 \quad \Rightarrow \quad II \geq \frac{l(\theta)}{d(\theta)}$$

Depth and Makespan

Depth

- Number of iterations alive in the kernel
- Secondary parameter
- Influence prolog/epilog size
- Complex relation with *II*

Makespan

- Time to complete a whole iteration in the kernel loop
- Secondary parameter
- Related to both *II* and depth of the pipeline
- Influence variable lifetimes

Software Pipelining Approaches

Main approaches

- Exact algorithms: enumerate every possibility (NP-Complete)
- Heuristics: best choice in production compilers

Heuristic family

- Modulo scheduling
- Kernel recognition

Today

- Approach mainly used in production compilers: *modulo scheduling*

Modulo Scheduling

Principle

- Schedule a single iteration such as it is valid repeted every *II* cycle
- Need to fix *II* before scheduling one iteration
- If not possible, then increase the value of targeted *II*

Main algorithm

```
Sort the nodes by priority
Compute MII
II = MII
While (schedule not valid)
    Schedule a single iteration with II
    If (schedule not valid)
        II++
```

Iterative Modulo Scheduling (IMS)

Principle

- Seminal work on modulo scheduling by Bob Rau, MICRO-27 (1994)
- Extension of list scheduling to loop
- Notion of budget

Compiler

- Implemented in Intel's compiler ICC

Iterative Modulo Scheduling (IMS)

- Pick up next instruction in decreasing priority H

$$H(P) = \begin{cases} 0 & \text{if P is a leaf} \\ \max_{Q \in Succ(P)} \left(H(Q) + L(P,Q) - II \times D(P,Q)\right) & \text{otherwise} \end{cases}$$

Iterative Modulo Scheduling (IMS)

- Pick up next instruction in decreasing priority H

$$H(P) = \begin{cases} 0 & \text{if P is a leaf} \\ \max_{Q \in Succ(P)} (H(Q) + L(P,Q) - II \times D(P,Q)) & \text{otherwise} \end{cases}$$

- Compute the range to schedule it: $[Estart, Estart + II - 1]$

$$Estart(P) = \max_{Q \in Pred(P)} \begin{cases} 0 & \text{if Q is unscheduled} \\ \max(0, \sigma(Q) + L(Q,P) - II \times D(Q,P)) & \text{otherwise} \end{cases}$$

Iterative Modulo Scheduling (IMS)

- Pick up next instruction in decreasing priority H

$$H(P) = \begin{cases} 0 & \text{if P is a leaf} \\ \max_{Q \in Succ(P)} (H(Q) + L(P,Q) - II \times D(P,Q)) & \text{otherwise} \end{cases}$$

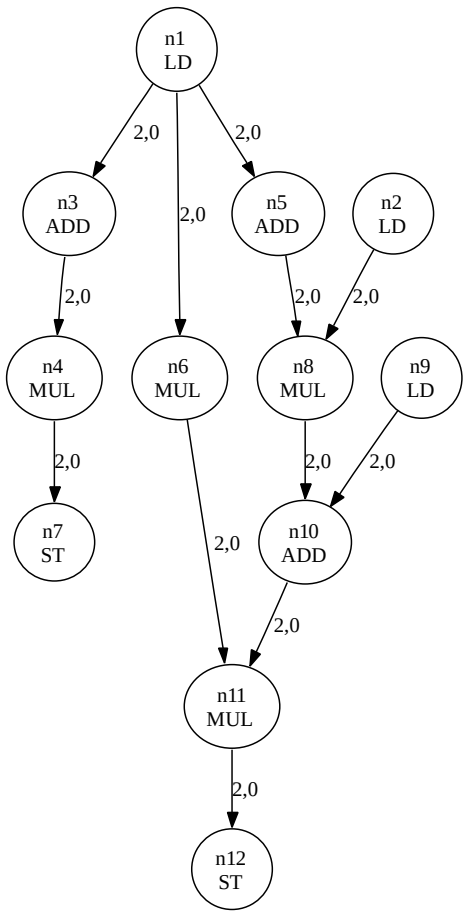
- Compute the range to schedule it: $[Estart, Estart + II - 1]$

$$Estart(P) = \max_{Q \in Pred(P)} \begin{cases} 0 & \text{if Q is unscheduled} \\ \max(0, \sigma(Q) + L(Q,P) - II \times D(Q,P)) & \text{otherwise} \end{cases}$$

- Try to schedule it within the range
- If failed (due to either data dependences or resource usage), then force the schedule and unschedule conflicting instructions
- Involve a notion of *budget* to avoid cyclically unscheduling the same set of instructions

IMS – Example 1 [LlosaPACT96]

DDG:



Reservation tables:

LD/ST	r0	r1	r2	r3
0			X	
1			X	

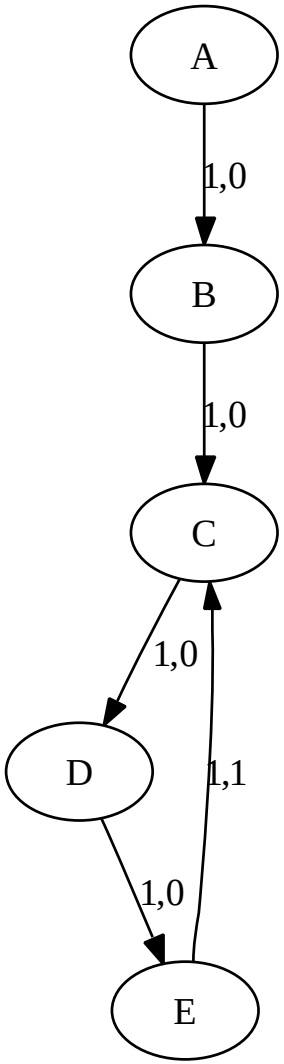
LD/ST	r0	r1	r2	r3
0				X
1				X

ADD	r0	r1	r2	r3
0	X			
1	X			

MUL	r0	r1	r2	r3
0		X		
1		X		

IMS – Example 2

DDG:



Reservation tables:

A	r0	r1	r2
0	X		

B	r0	r1	r2
0	X		

C	r0	r1	r2
0		X	

D	r0	r1	r2
0	X		

E	r0	r1	r2
0			X

IMS – Example 2

1. Compute MII:
   - RecMII = 3, ResMII = 3 $\Rightarrow$ MII = 3
2. Compute priority H
   - $H(A) = 4$, $H(B) = 3$, $H(C) = 2$, $H(D) = 1$, $H(E) = 0$
3. Start the scheduling process with $II = MII = 3$

Cycle	Schedule
0	A
1	B
2	C
3	
4	
5	D
6	E

Cycle	r0	r1	r2
0	X		X
1	X		
2	X	X	

4. Success: $II = 3$, $D = 3$, $M = 7$

Swing Modulo Scheduling (SMS)

Principle

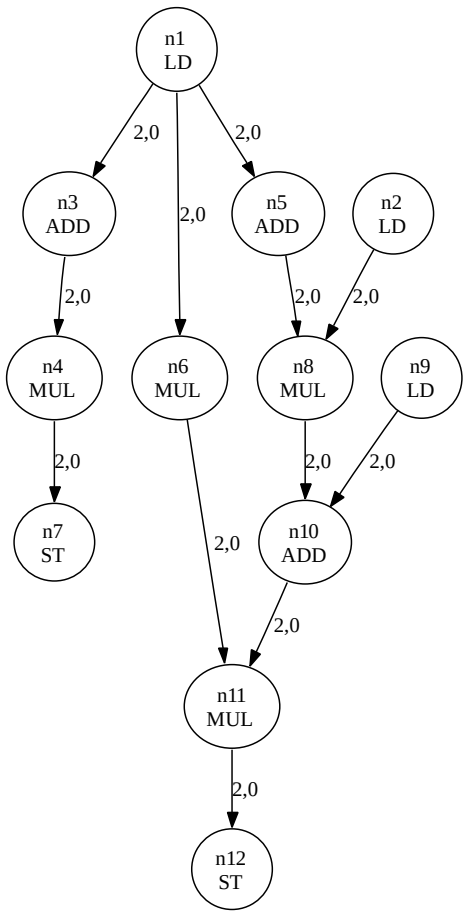
- By Llosa *et al.*, PACT'96
- Avoid the need to unscheduled instructions
  - i.e., when both predecessors and successors are scheduled
- Based on the scheduling of strongly connected components (SCC)
  - Sort SCC by decreasing RecMII
- Go backward and forward on nodes linking two SCCs
- When it is impossible to schedule an instruction, do not force, just increase *II*

Compiler

- Implemented in GCC by IBM Haifa

SMS – Example 1 [LlosaPACT96]

DDG:



Reservation tables:

LD/ST	r0	r1	r2	r3
0			X	
1			X	

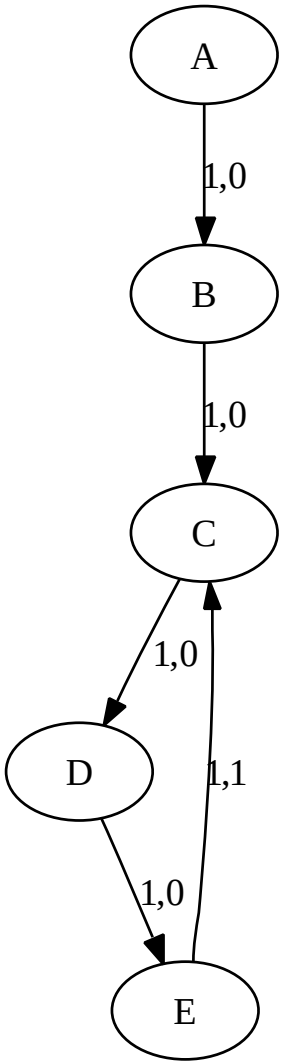
LD/ST	r0	r1	r2	r3
0				X
1				X

ADD	r0	r1	r2	r3
0	X			
1	X			

MUL	r0	r1	r2	r3
0		X		
1		X		

SMS – Example 2

DDG:



Reservation tables:

A	r0	r1	r2
0	X		

B	r0	r1	r2
0	X		

C	r0	r1	r2
0		X	

D	r0	r1	r2
0	X		

E	r0	r1	r2
0			X

SMS – Example 2

1. Compute MII:
   - RecMII = 3, ResMII = 3 $\Rightarrow$ MII = 3
2. Compute priority order O
   - $O =< C, D, E, B, A >$
3. Start the scheduling process with $II = MII = 3$

Cycle	Schedule
-3	A
-2	
-1	B
0	C
1	D
2	E

Cycle	r0	r1	r2
0	X	X	
1	X		
2	X		X

4. Success: $II = 3$, $D = 2$, $M = 6$

Code Generation and Hardware Support

Register Allocation

- Classical register allocation
- Problems arise when lifetimes exceed *II* cycles
- Solutions:
  - Software: Modulo variable expansion
  - Hardware: Rotating register file

Predication

- Small irregular control $\Rightarrow$ If-conversion
- Kernel-only loop

Example of architecture

- Overview of Itanium architecture

Register Allocation

Example

```
for ( i=0; i<N; i++ ) // Latencies of 2 cycles
  | Z[i] = X[i] + Y[i]; // @X in r2, @Y in r3, @Z in r4
```

- Software-pipelined schedule with $II = 1$:

Cycle	Schedule				
0	ST[r4]=r7,8		FMA r7=r5,r1,r6		LD r5=[r2],8 LD r6=[r3],8

- Lifetime of 2 cycles $\Rightarrow$ produced values are erased

Solutions

1. Unroll the kernel 2 times (Modulo Variable Expansion)
2. Use rotating registers (hardware feature)

Predication

Principle

- Convert control dependence into data dependence
- Add one bit (predicate) per instruction
- Instruction is commited iff the predicate is true

Applications

- If-conversion (small control irregularity)
- Kernel-only schedule

Example:

```
if (c)
  | A ;
else
  | B ;
```

$\Rightarrow$

```
      cmp p1,p2=c
(p1) A;
(p2) B;
```

Itanium 2 Architecture

EPIC (Explicitly Parallel Instruction Computing)

- VLIW-like architecture except for memory operations (out-of-order memory accesses)
- IPC up to 6 (Instructions Per Cycle)
- Compiler has to expose parallelism (+ constraints on instruction grouping: *bundling*)
- Provides fully support for software-pipelined schedules: rotating register files and predication

Currently

- Itanium2 Montecito
- Dual-core with hyperthreading (2-way)
- Seperate caches: L1, L2 and L3 (12MB for L3)

Conclusion

Instruction Scheduling

- Input: set of instructions
- Output: order on instructions
- Must respect both data dependences and resource constraints
  - Need a model to represent such constraints (DDG, reservation tables, automaton, ...)

Schedule Types

- Basic block: List scheduling
- Loop:
  - List scheduling on the body $\Rightarrow$ do not benefit from intra-iteration parallelism
  - Software pipelining $\Rightarrow$ Modulo scheduling heuristics (IMS, SMS)