Hypernode Reduction
Modulo Scheduling

J. Llosa, M. Valero, E. Ayguadé and A. Gonzalez

Dept. d'Arquitectura de Computadors

Universitat Politècnica de Catalunya

MICRO-28

28th Annual International Symposium on Microarchitecture

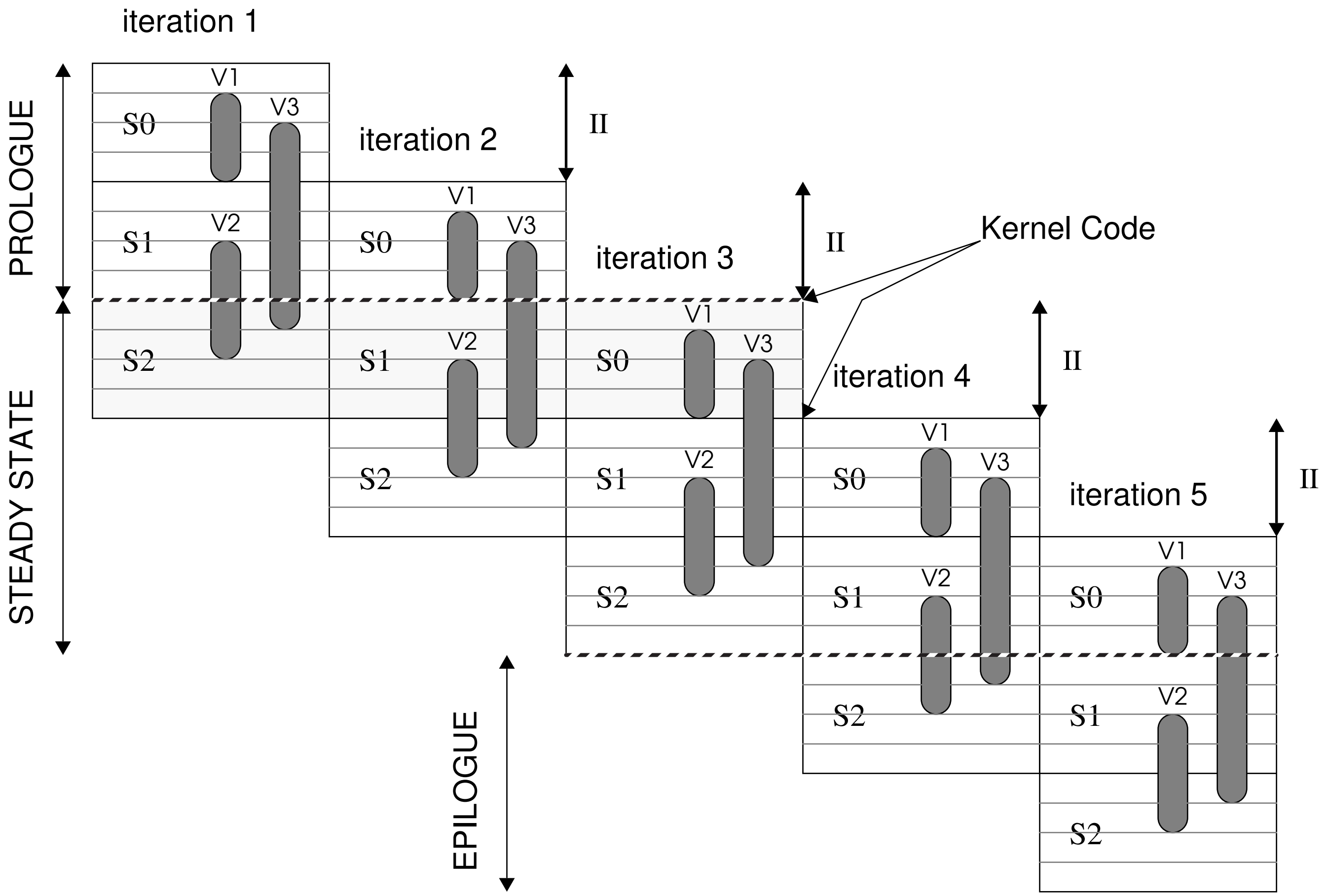
November 29-December 1, 1995

Ann Arbor, Michigan, USA

Talk Outline

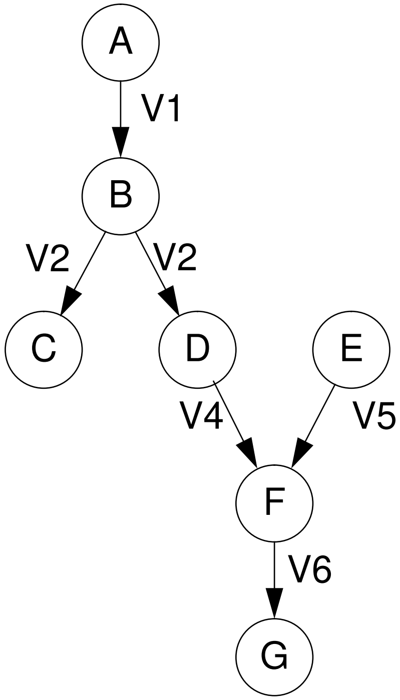
- **Introduction**
- **Motivating Example**
- **Hypernode Reduction Modulo Scheduling**
  - Pre-Ordering Step Without Recurrence Circuits
  - Scheduling Step
  - Pre-Ordering Step With Recurrence Circuits
  - Scheduling Step
- **Experimental Results**
- **Conclusions**

Introduction

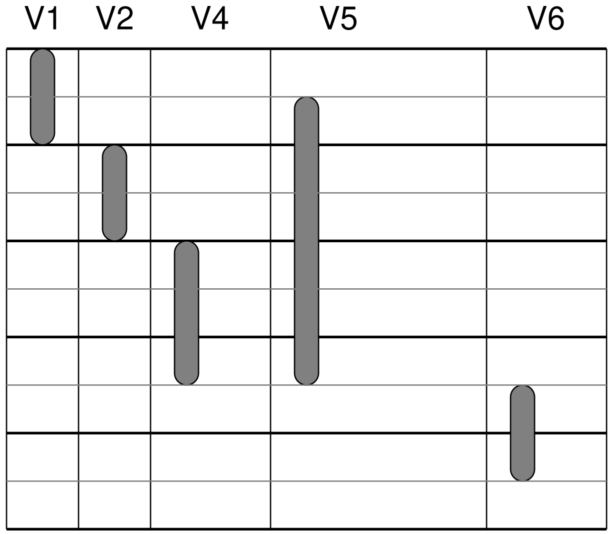


Motivating Example

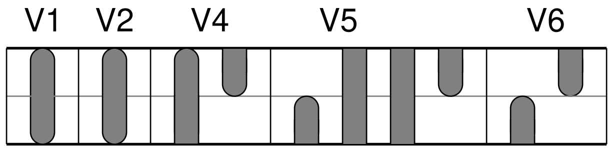
Top-Down Scheduling



Cycle				
0	A			
1	E			
2		B		
3				
4			C	D
5				
6				
7		F		
8				
9			G	

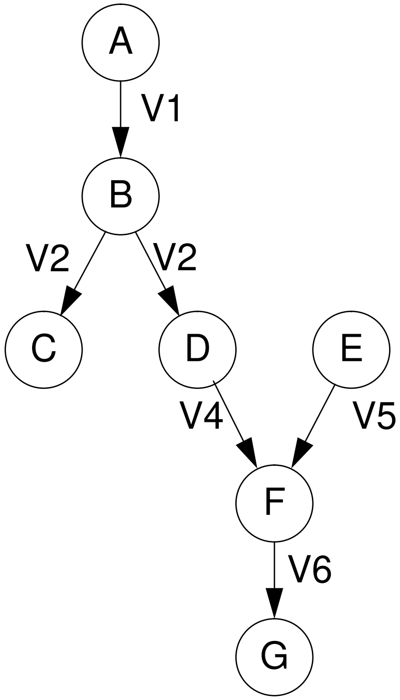


0	A_0	B_1	C_2	D_2
1	E_0	F_3	G_4	

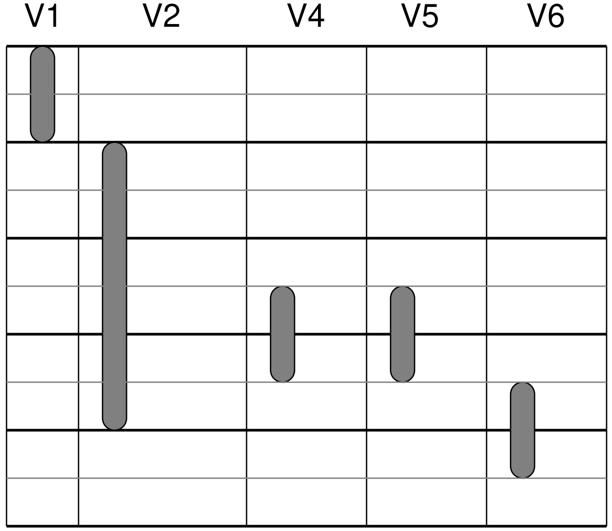


Motivating Example

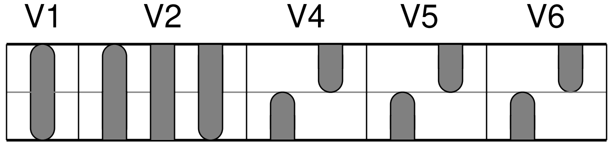
Bottom-Up Scheduling



Cycle				
0	A			
1				
2		B		
3				
4				
5			E	D
6				
7		F		
8			C	
9	G			

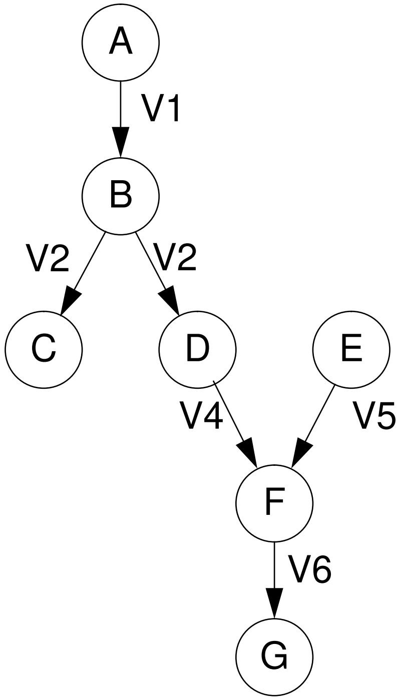


0	A_0	B_1	C_4	
1	G_4	F_3	E_2	D_2

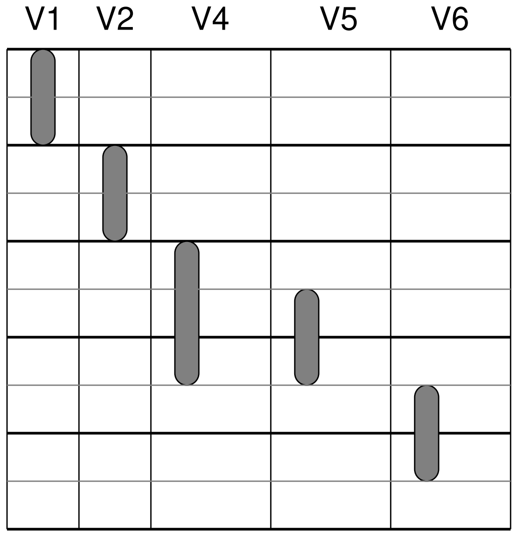


Motivating Example

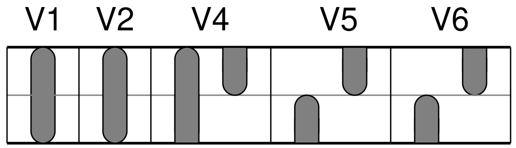
Bidirectional Scheduling



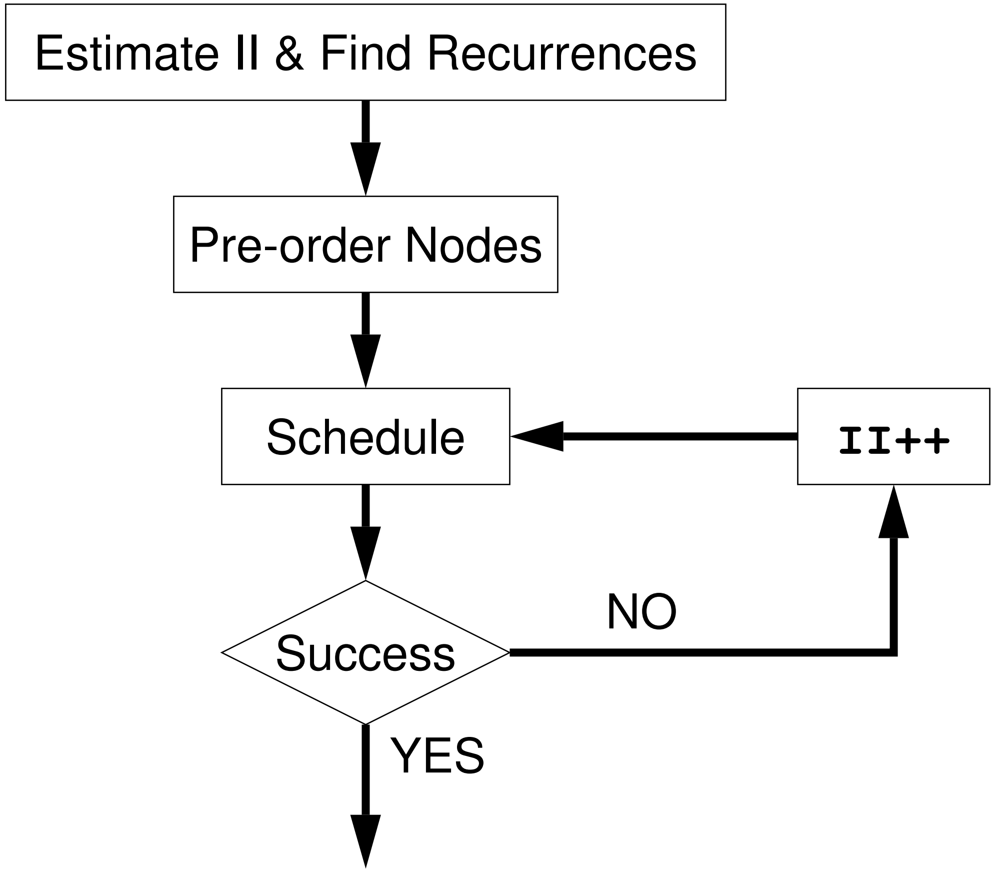
Cycle				
0	A			
1				
2		B		
3				
4			C	D
5		E		
6				
7	F			
8				
9			G	



0	A_0	B_1	C_2	D_2
1	F_3	E_2	G_4	



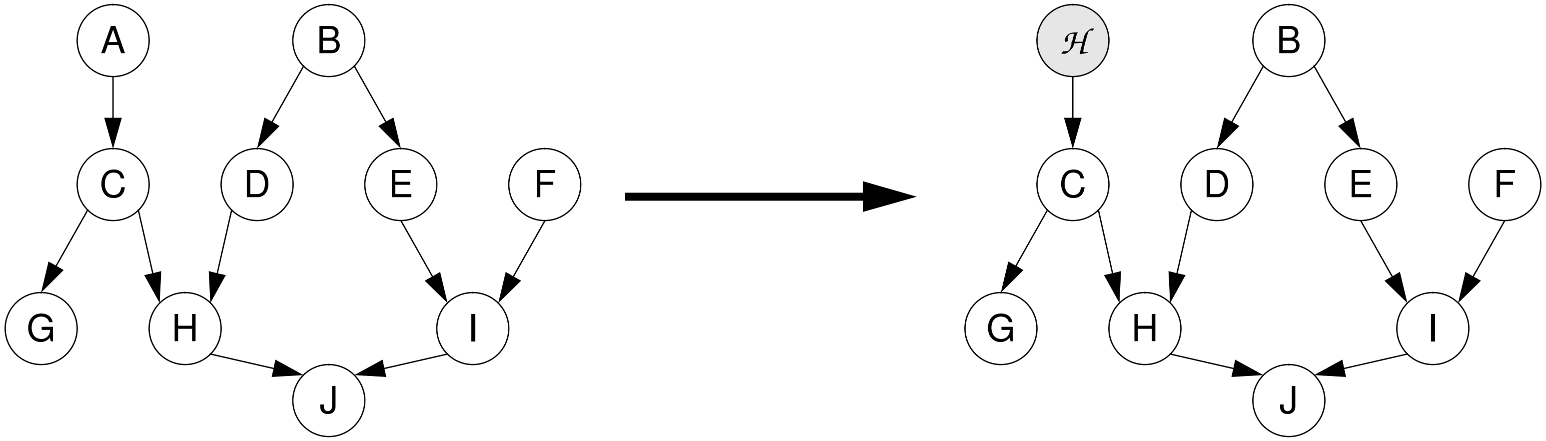
Hypernode Reduction Modulo Scheduling



Pre-Ordering without Recurrences

- **Select a node as the Hypernode**
- **While there are nodes to be ordered**
  - Select the predecessors of the Hypernode
  - Find the nodes in any path between predecessors
  - Topologically order the selected nodes
  - Insert the nodes in the ordered list
  - Reduce all the selected nodes to the Hypernode
  - IDEM for the successors

Pre-Ordering without Recurrences



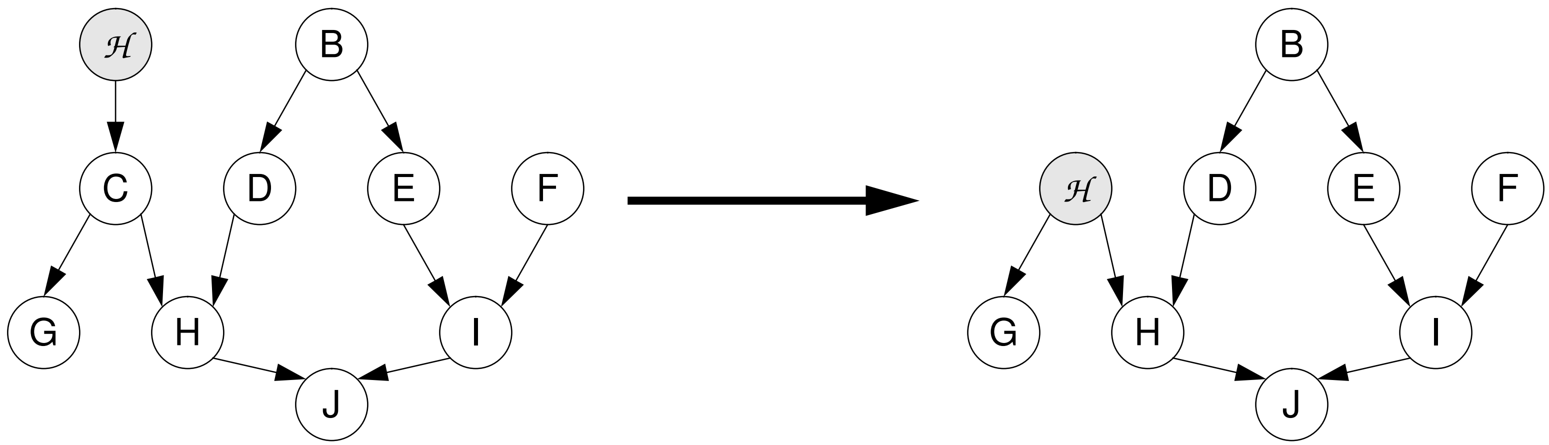
☞**Select a node as the Hypernode**

➥List = { A }

CEPBA

DAC

Pre-Ordering without Recurrences



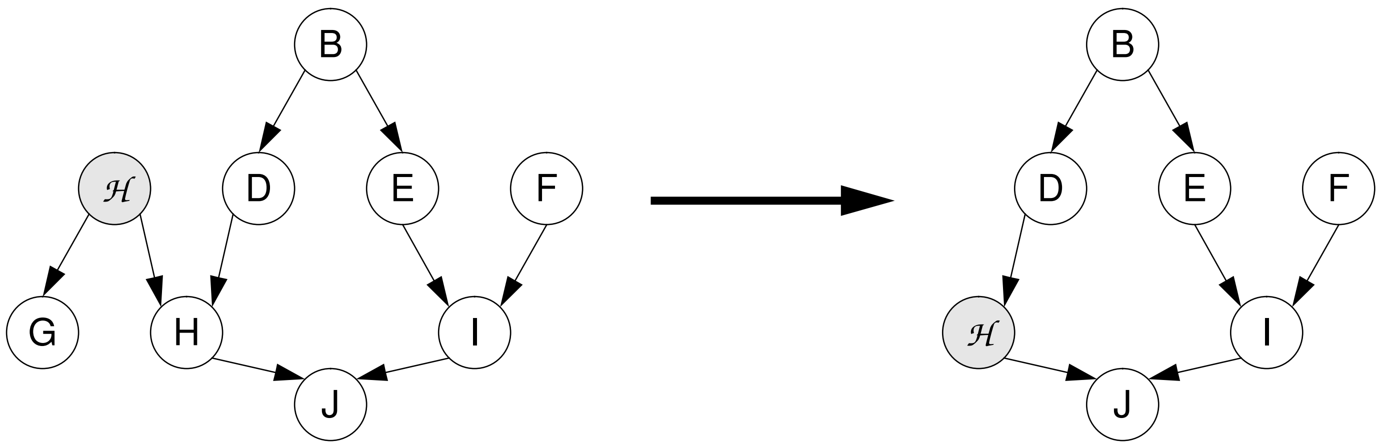
☞**Select the successors of $\mathcal{H}$: { C }**

➥Nodes in any path: { C }

➥Topological sort: { C }

➥List = { A, C }

Pre-Ordering without Recurrences



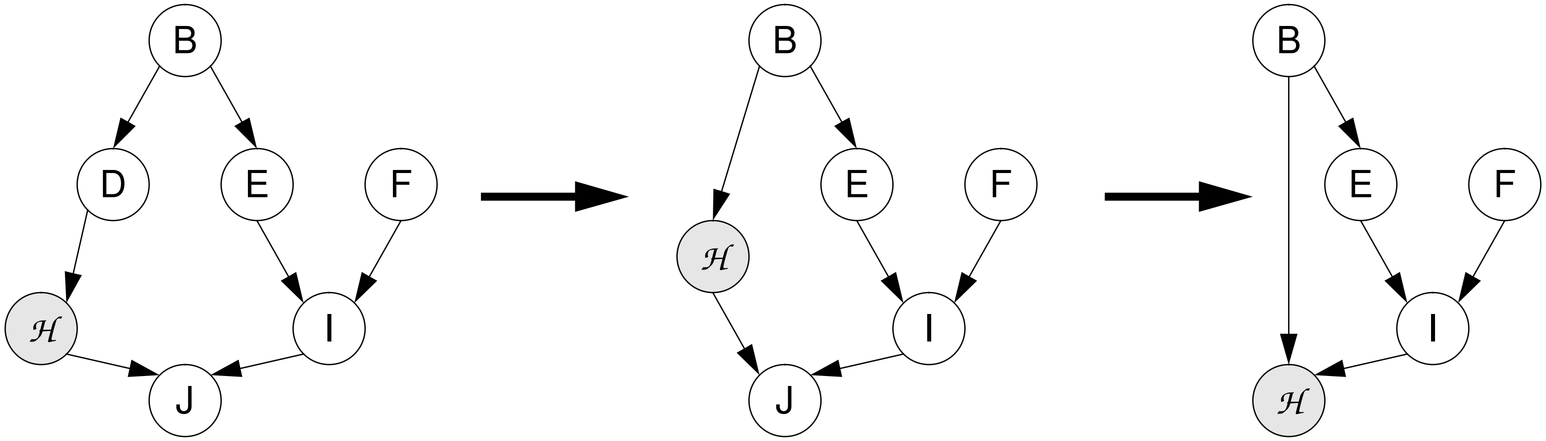
☞**Select the successors of $\mathcal{H}$: { G, H }**

➥Nodes in any path: { G, H }

➥Topological sort (ASAP): { G, H } (in fact {H, G } will be also valid)

➥List = { A, C, G, H }

Pre-Ordering without Recurrences



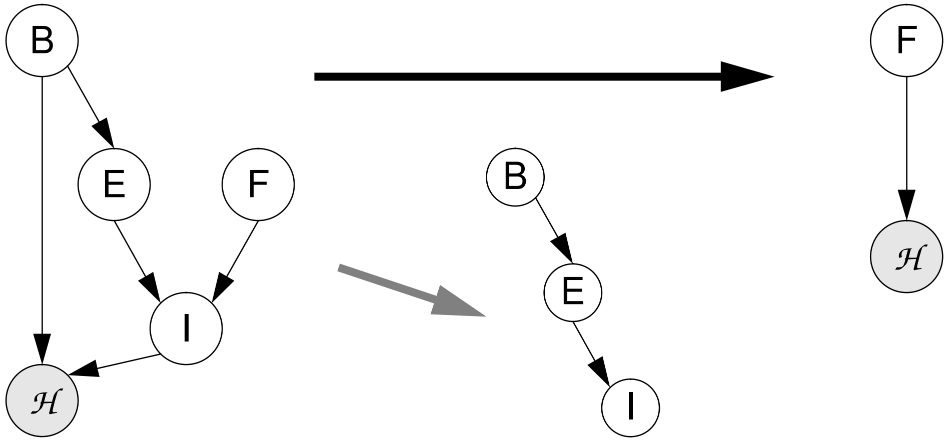
☞**Select the predecessors of $\mathcal{H}$: { D }**

➥List = { A, C, G, H, D }

☞**Select the successors of $\mathcal{H}$: { J }**

➥List = { A, C, G, H, D, J }

Pre-Ordering without Recurrences



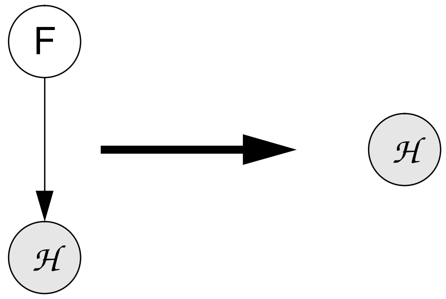
☞**Select the predecessors of $\mathcal{H}$: { B, I }**

- ➥Nodes in any path: { B, E, I }
- ➥Topological sort (Inverted ALAP): { I, E, B }
- ➥List = { A, C, G, H, D, J, I, E, B }

CEPBA

D A C

Pre-Ordering without Recurrences



☞**Select the predecessors of $\mathcal{H}$: { F }**

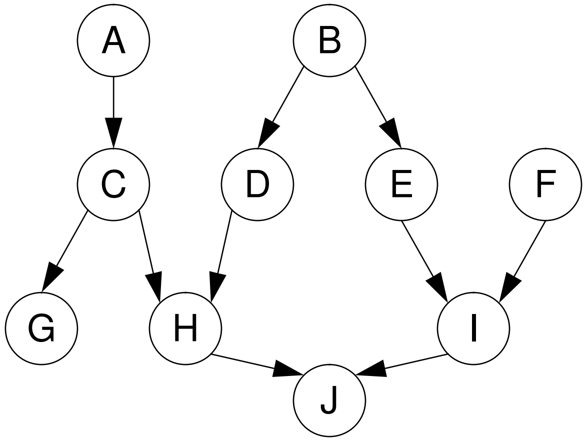
➥List = { A, C, G, H, D, J, I, E, B, F }

Scheduling Step

☞ **For each node**

- ➥ If the node has predecessors in the partial schedule
  - ➢ Schedule the node as soon as posible
- ➥ If the node has successors in the partial schedule
  - ➢ Schedule the node as late as posible

Scheduling Step



List = { A, C, G, H, D, J, I, E, B, F }

A_0		

A_0		
C_0		

A_0	G_1	
C_0		

A_0	G_1	H_1
C_0		

A_0	G_1	H_1
C_0	D_0	

A_0	G_1	H_1
C_0	D_0	J_1

A_0	G_1	H_1
C_0	D_0	J_1
I_0		

A_0	G_1	H_1
E_0		
C_0	D_0	J_1
I_0		

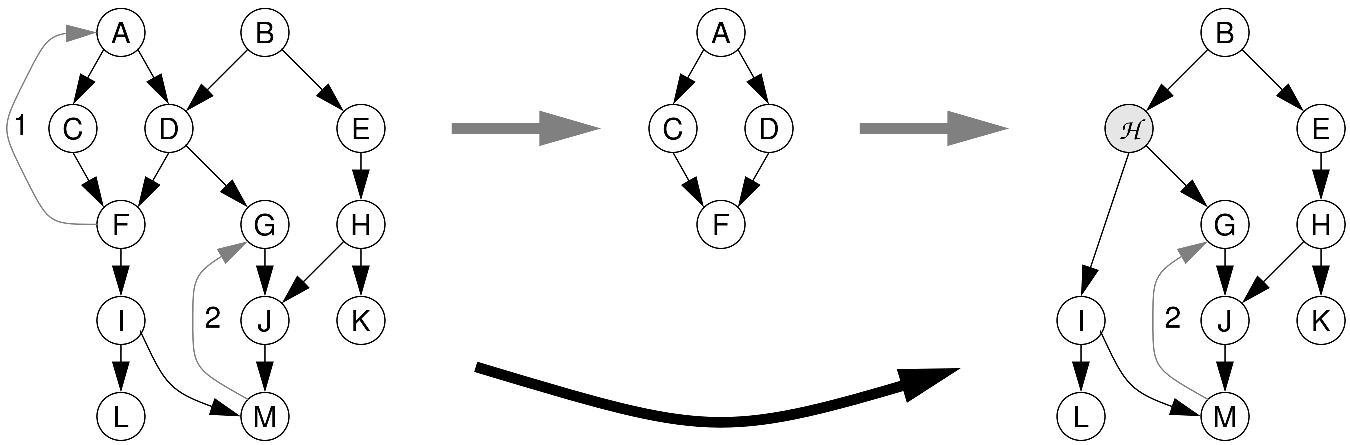
A_0	G_1	H_1
E_0		
C_0	D_0	J_1
I_0	B_{-1}	

A_0	G_1	H_1
E_0	F_0	
C_0	D_0	J_1
I_0	B_{-1}	

Pre-Ordering with Recurrences

- **Select the most restrictive recurrence subgraph**
- **Apply pre-ordering without recurrences to the recurrence subgraph**
- **While there are recurrence subgraphs**
  - Select next recurrence subgraph
  - Find all paths from/to the Hypernode
  - Apply pre-ordering without recurrences
- **Apply Pre-ordering without recurrences to the resulting graph**

Pre-Ordering with Recurrences

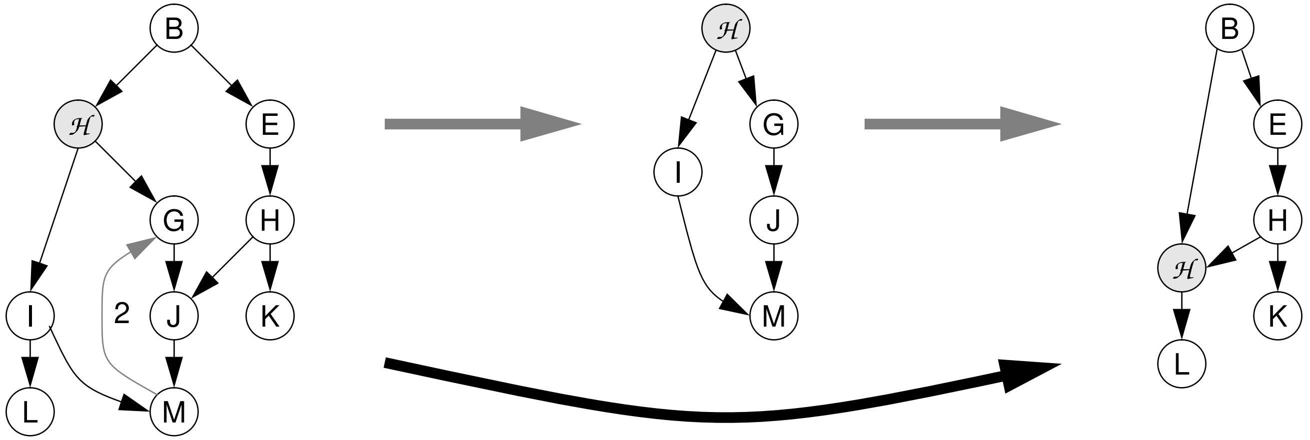


☞ **Select the most restrictive recurrence subgraph: { A, C, D, F }**

➥Apply Pre-ordering without recurrences: {A, C, D, F }

➥List = {A, C, D, F }

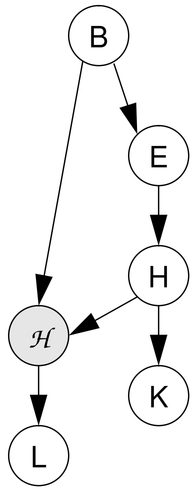
Pre-Ordering with Recurrences



☞**Select the most restrictive recurrence subgraph: { G, J, M }**

- ➥Find all paths between $\mathcal{H}$ and { G, J, M } : { $\mathcal{H}$, I, G, J, M }
- ➥Apply Pre-ordering without recurrences: { I, G, J, M }
- ➥List = {A, C, D, F, I, G, J, M }

Pre-Ordering with Recurrences



☞**If there are no more recurrence subgraphs:**

- ➥Apply Pre-ordering without recurrences: { H, E, B, L, K }
- ➥List = {A, C, D, F, I, G, J, M, H, E, B, L, K }

Scheduling Step

- **For each node**
  - If the node has predecessors in the partial schedule
    - Schedule the node as soon as posible
  - If the node has successors in the partial schedule
    - Schedule the node as late as posible
  - If the node has predecessors and successors
    - Schedule the node in the first free slot scanning from the last predecessor to the first successor
- **If no free slot is encountered re-schedule all the nodes with an increased II**

Experimental Results

☞ **Comparison with other methods:**

- ➥ SPILP: Optimal integer linear programming method that minimizes register requirements
- ➥ Slack: Heuristic method that tries to reduce the register requirements
- ➥ FRLC: Heuristic method that doesn't consider register requirements

☞ **Loops selected from: SPEC, Livermore, Linpack, Whetstone**

Experimental Results

☞Performance

HRMS	II <	II =			II >
		Buffers <	Buffers =	Buffers >	
SPILP	0	0	24	0	0
Slack	7	1	16	0	0
FRLC	9	8	7	0	0

☞Time to construct the schedules

	HRMS	HRMS (paper)	SPILP	Slack	FRLC
Time	0.32	1.45	290.72	0.93	0.71

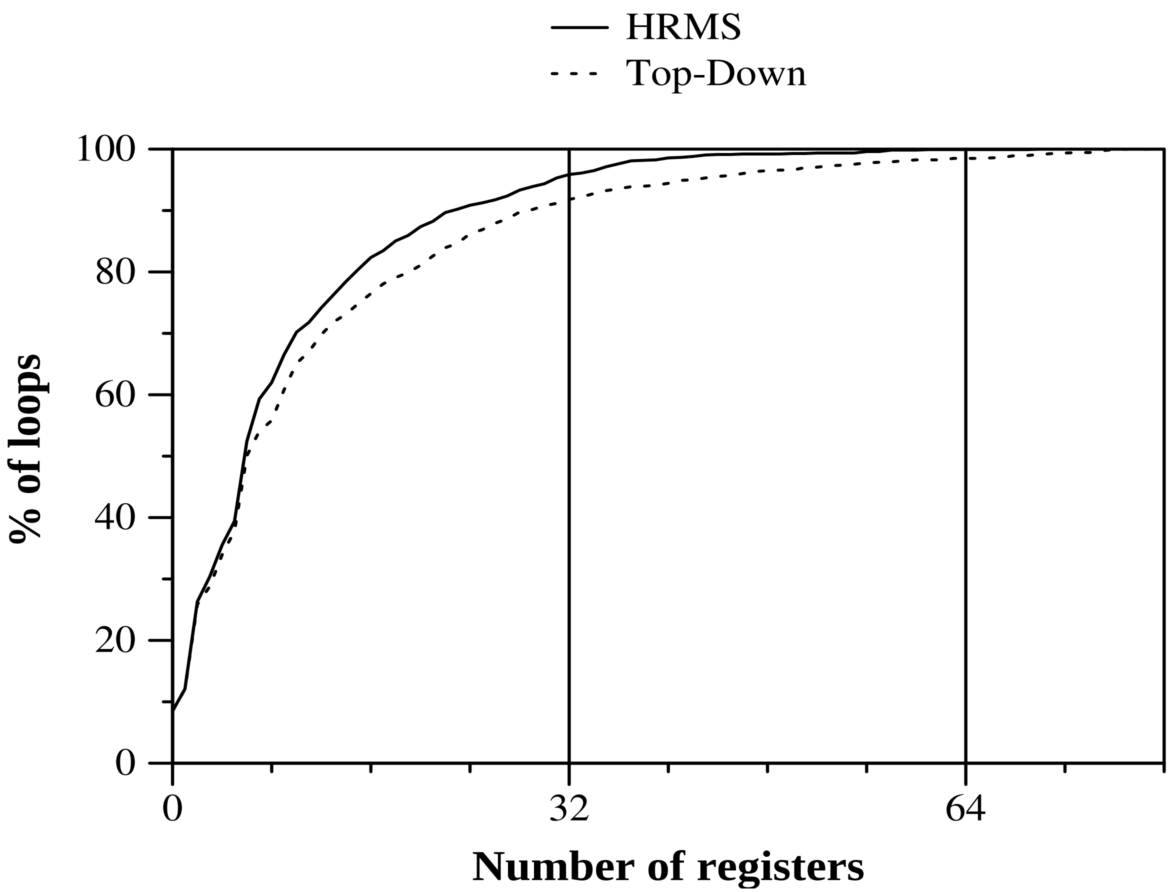
Experimental Results

☞Perfect Club Loops

- ➥1258 loops → 78 % Execution time
  - ➢ 152 (12 %) have Div/Mod/Sqrt
  - ➢ 505 (40 %) have Recurrences
- ➥ Scheduling time: 55 seconds
  - ➢ 3.6 (7 %) Search recurrences
  - ➢ 36.5 (66 %) Ordering step
  - ➢ 15.0 (27 %) Scheduling step
- ➥ Performance
  - ➢ 31 (2.5 %) loops with II > MII
  - ➢ II = 1.013 * MII

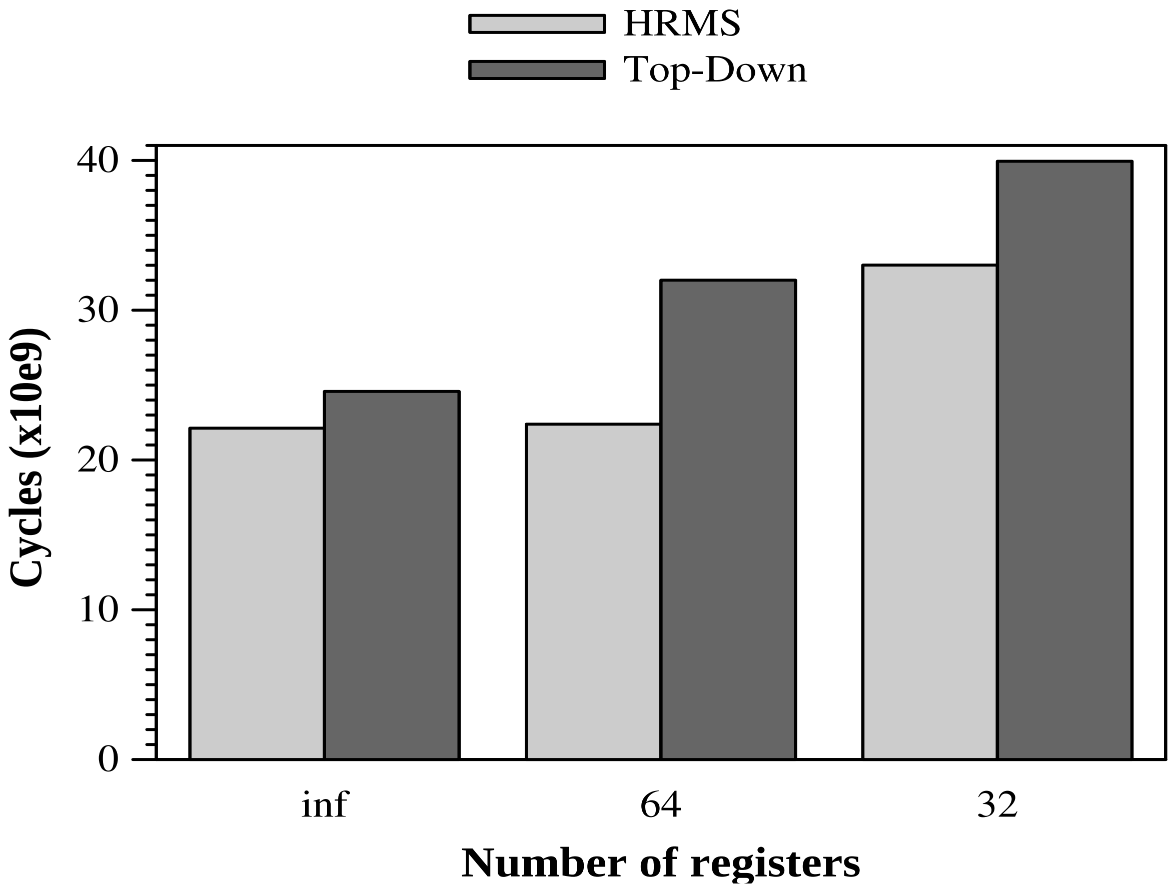
Experimental Results

☞ **Register requirements of loop-variants**



Experimental Results

☞**Performance with fixed size register files**



Conclusions

- **Hypernode Reduction Modulo Scheduling**
  - Static Priority Function: Hypernode reduction
  - Simple Scheduling: bidirectional greedy modulo scheduling
- **Reduces the lifetime of values**
  - Reduces the register requirements
- **Performance**
  - Almost optimal schedules (II = MII)
  - Short scheduling time