LUCAS: Latency-adaptive Unified Cluster Assignment and instruction Scheduling

Vasileios Porpodas and Marcelo Cintra

University of Edinburgh

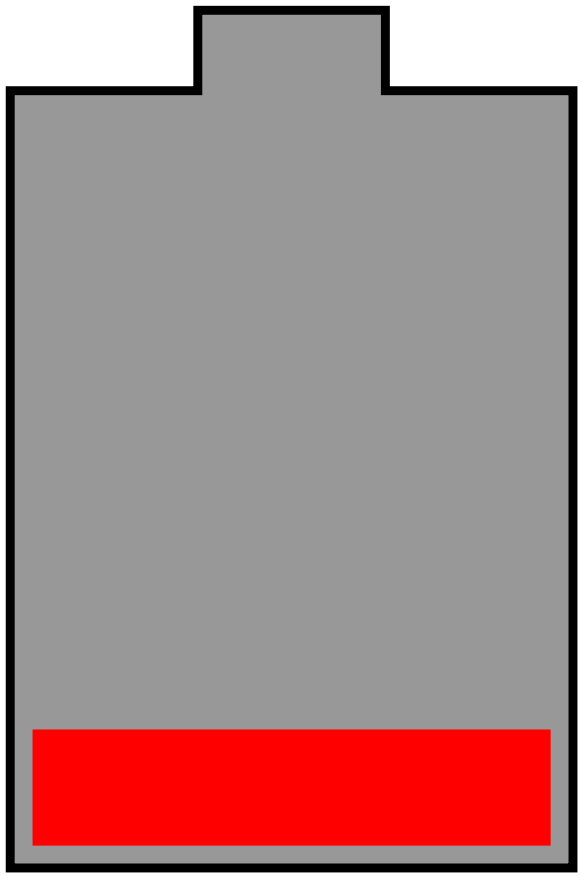
LCTES 2013

Energy efficiency, VLIWs and Scheduling

- Mobile era.

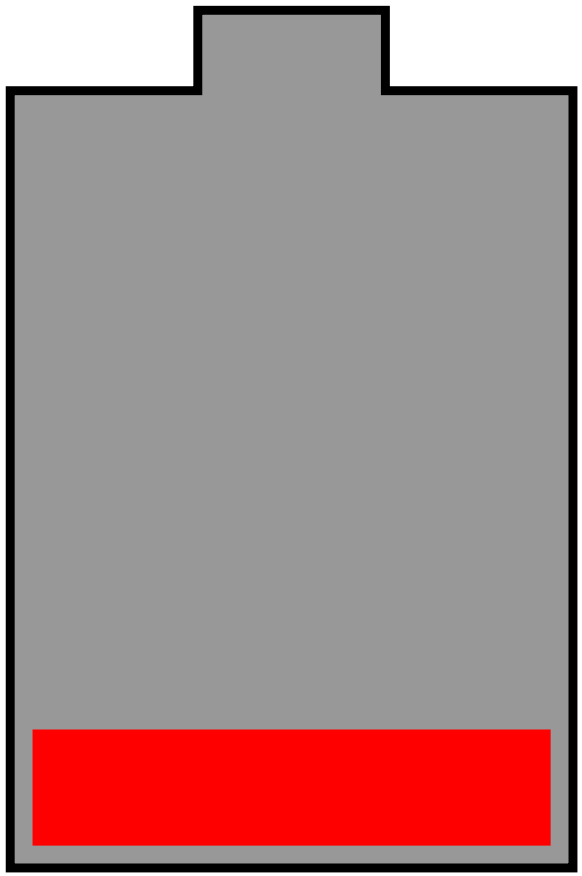
Energy efficiency, VLIWs and Scheduling

- Mobile era.
- Energy becomes a major design constraint.



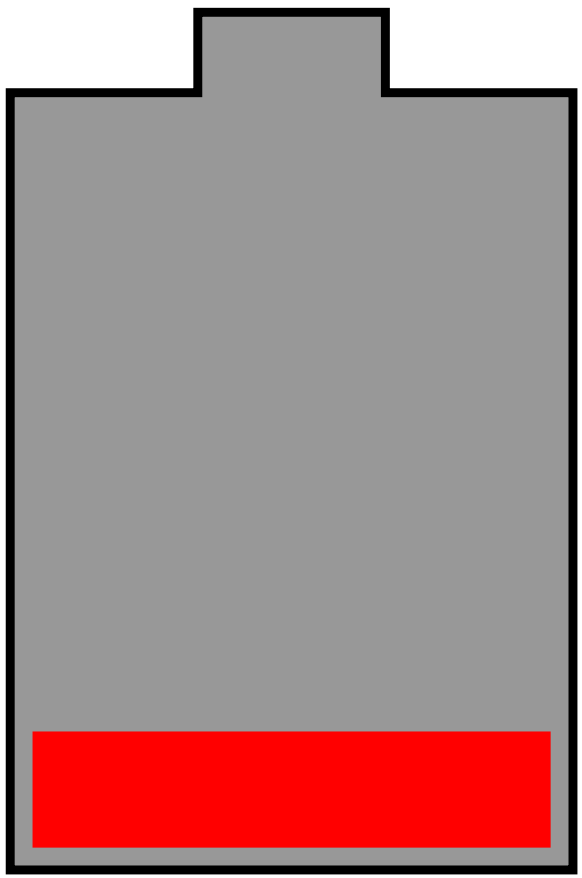
Energy efficiency, VLIWs and Scheduling

- Mobile era.
- Energy becomes a major design constraint.
- Hardware Instruction scheduling consumes a lot of energy.



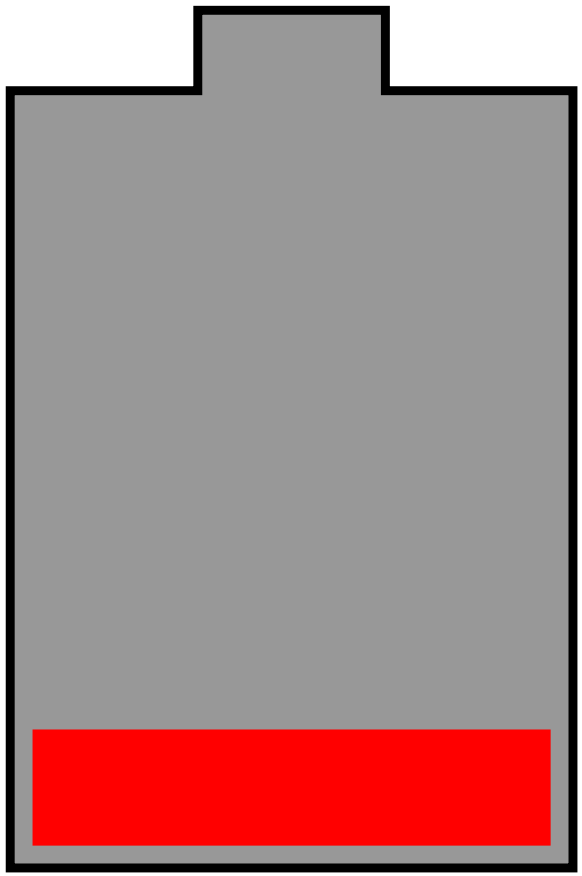
Energy efficiency, VLIWs and Scheduling

- Mobile era.
- Energy becomes a major design constraint.
- Hardware Instruction scheduling consumes a lot of energy.
- VLIW processors: high-performance and statically scheduled.



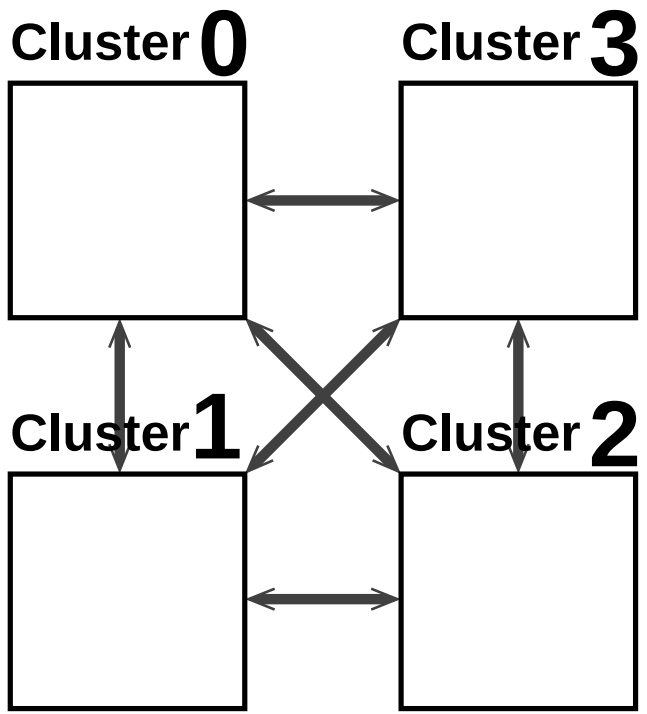
Energy efficiency, VLIWs and Scheduling

- Mobile era.
- Energy becomes a major design constraint.
- Hardware Instruction scheduling consumes a lot of energy.
- VLIW processors: high-performance and statically scheduled.
- Clustered VLIW = scalable VLIW.



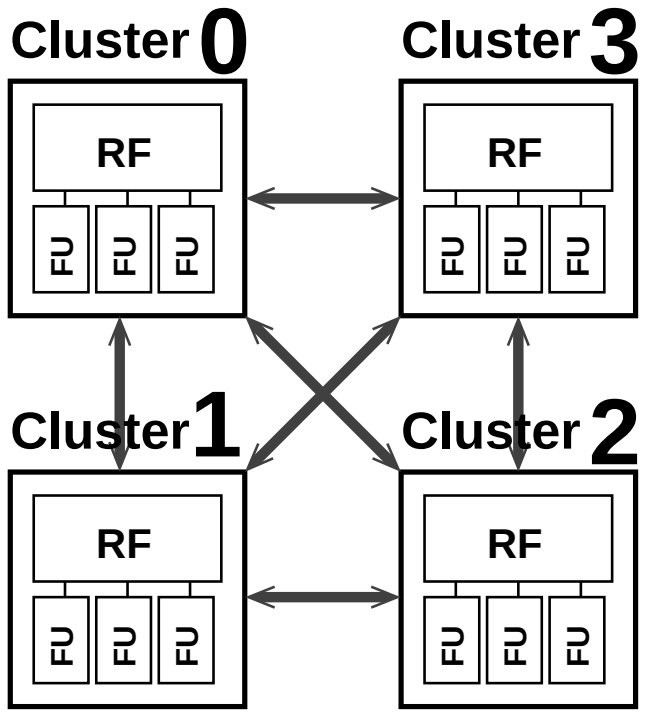
Clustered VLIW

- Scalable
- Energy efficient



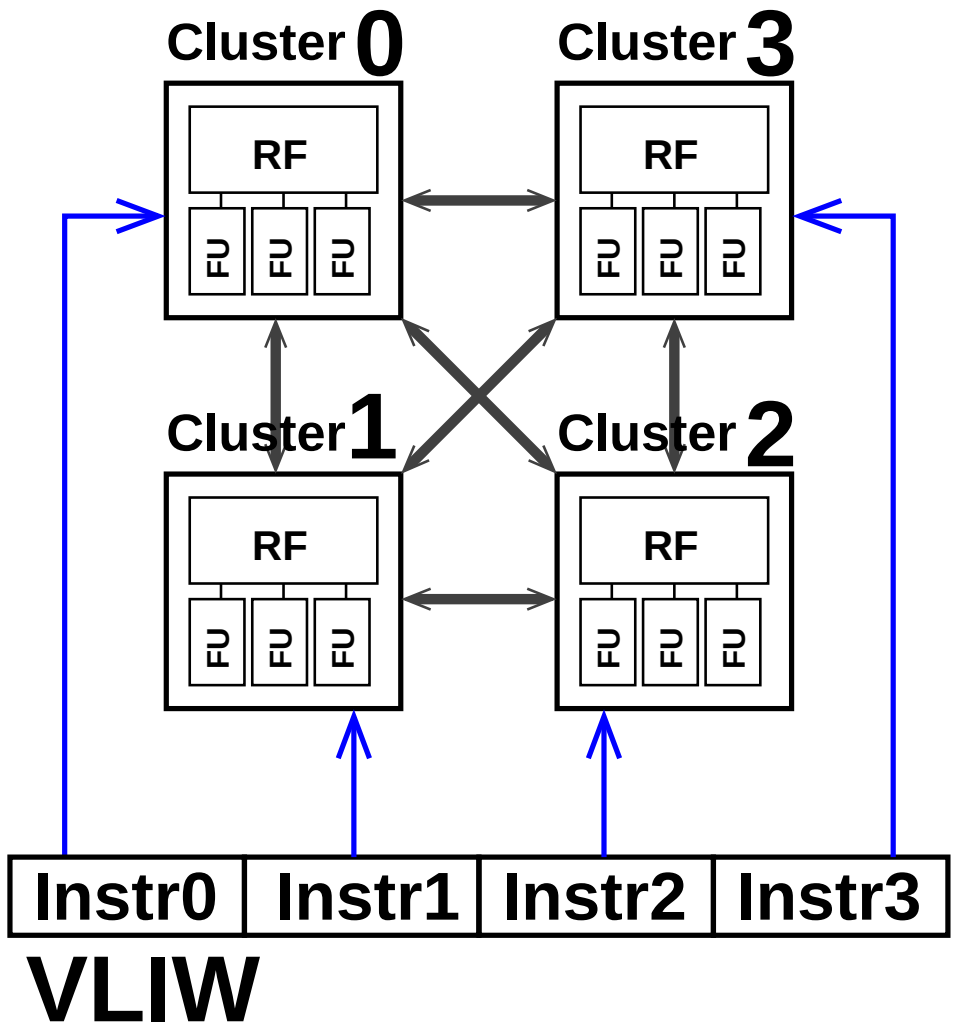
Clustered VLIW

- Scalable
- Energy efficient
- High frequency
- Inter-Cluster delay



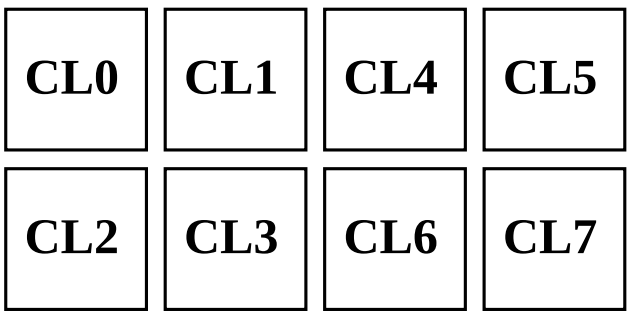
Clustered VLIW

- Scalable
- Energy efficient
- High frequency
- Inter-Cluster delay
- Relies on compiler
- Statically scheduled
- Explicit ILP



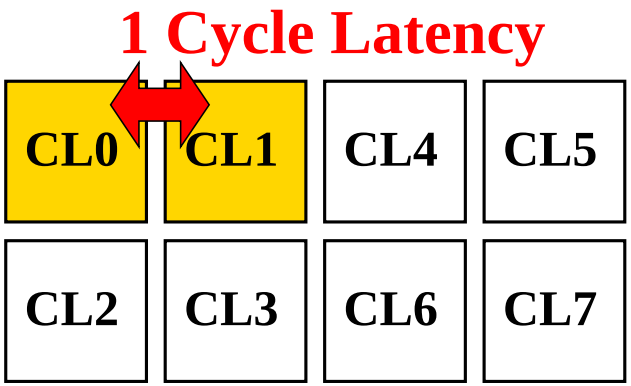
Problem: Many possible Inter-cluster latencies

- An application could map on any set of clusters.



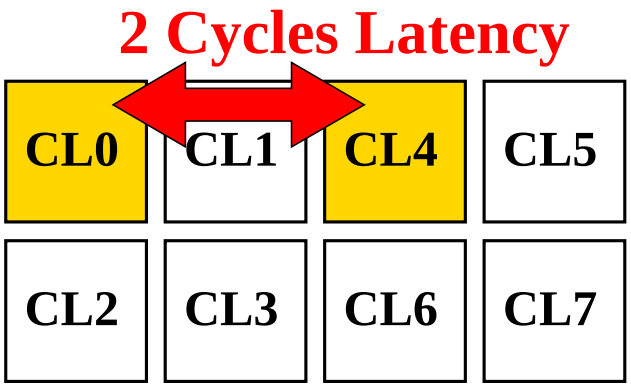
Problem: Many possible Inter-cluster latencies

- An application could map on any set of clusters.
- Low inter-cluster latency.



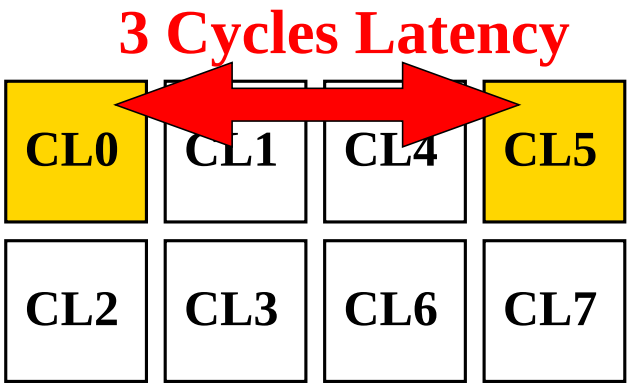
Problem: Many possible Inter-cluster latencies

- An application could map on any set of clusters.
- Low inter-cluster latency.
- Higher inter-cluster latency.



Problem: Many possible Inter-cluster latencies

- An application could map on any set of clusters.
- Low inter-cluster latency.
- Higher inter-cluster latency.
- Instruction Scheduler should adapt to all latencies.

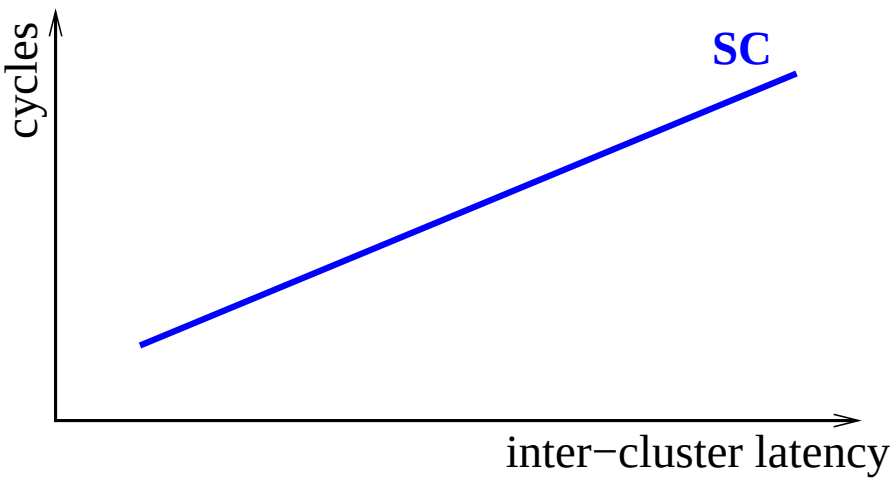


Problem Definition

- Existing schedulers perform well only:

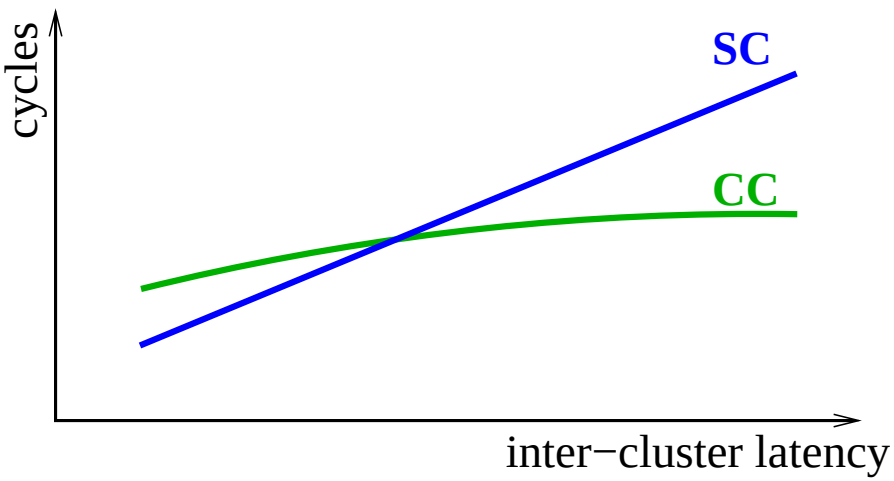
Problem Definition

- Existing schedulers perform well only:
- under low inter-cluster latency (SC),



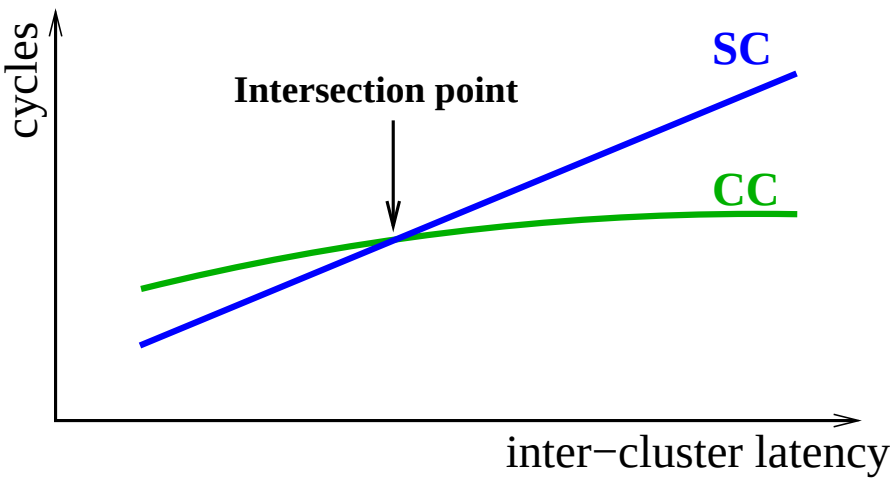
Problem Definition

- Existing schedulers perform well only:
- under low inter-cluster latency (SC),
- under high latency (CC).



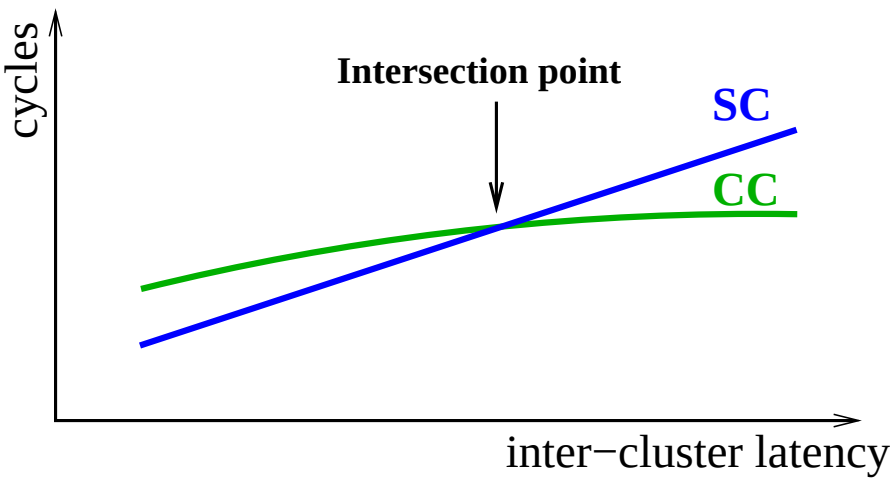
Problem Definition

- Existing schedulers perform well only:
- under low inter-cluster latency (SC),
- under high latency (CC).
- When each performs best is benchmark dependent.



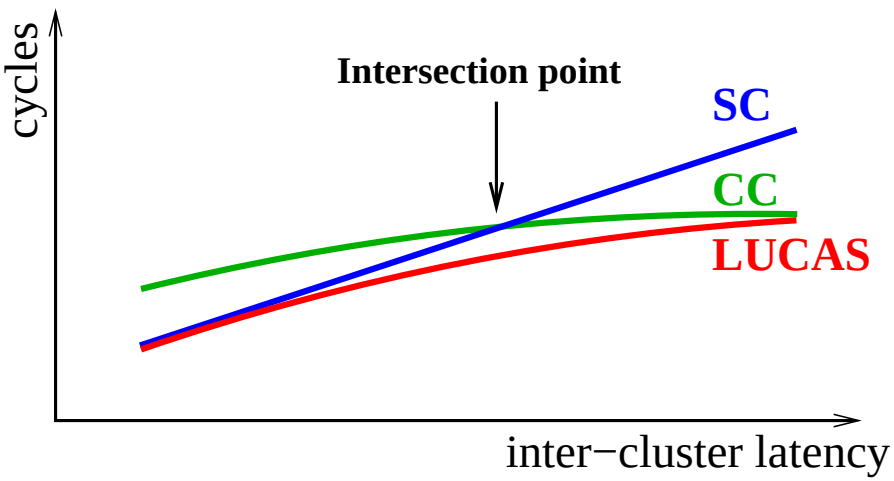
Problem Definition

- Existing schedulers perform well only:
- under low inter-cluster latency (SC),
- under high latency (CC).
- When each performs best is benchmark dependent.



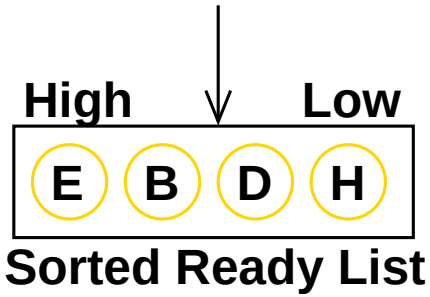
Problem Definition

- Existing schedulers perform well only:
- under low inter-cluster latency (SC),
- under high latency (CC).
- When each performs best is benchmark dependent.
- Scheduler must generate good code under any inter-cluster latency.

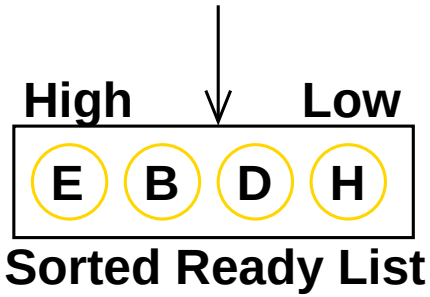


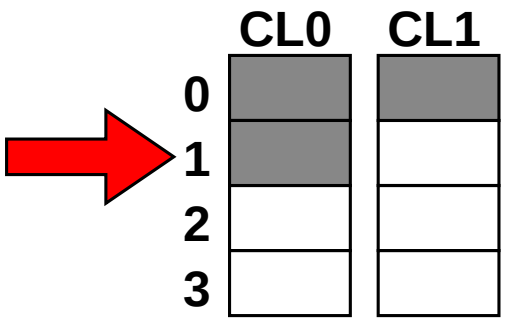
Outline

LUCAS Scheduler

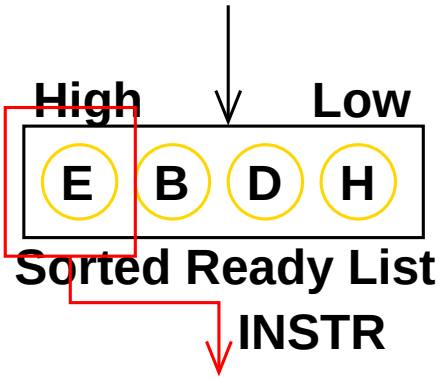


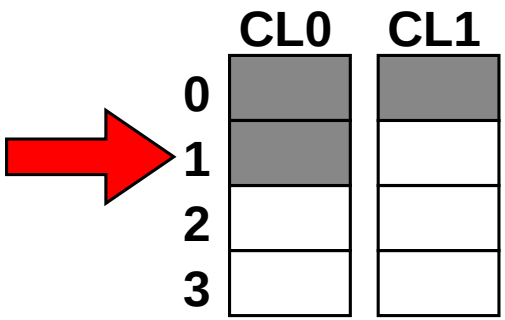
LUCAS Scheduler



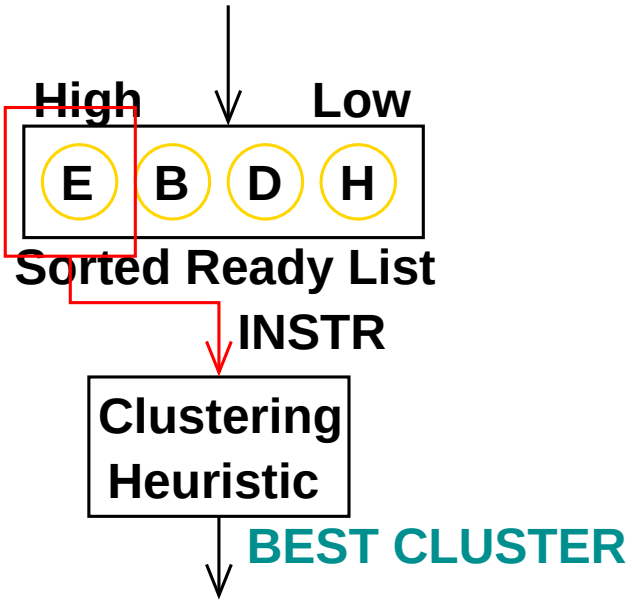


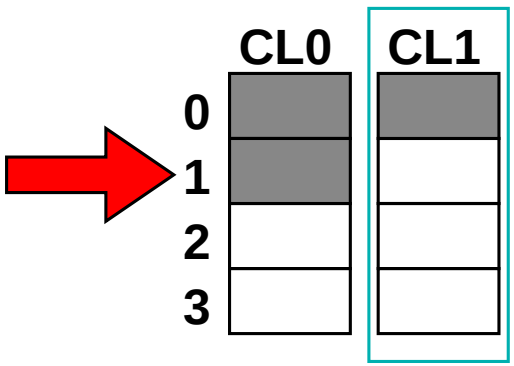
LUCAS Scheduler



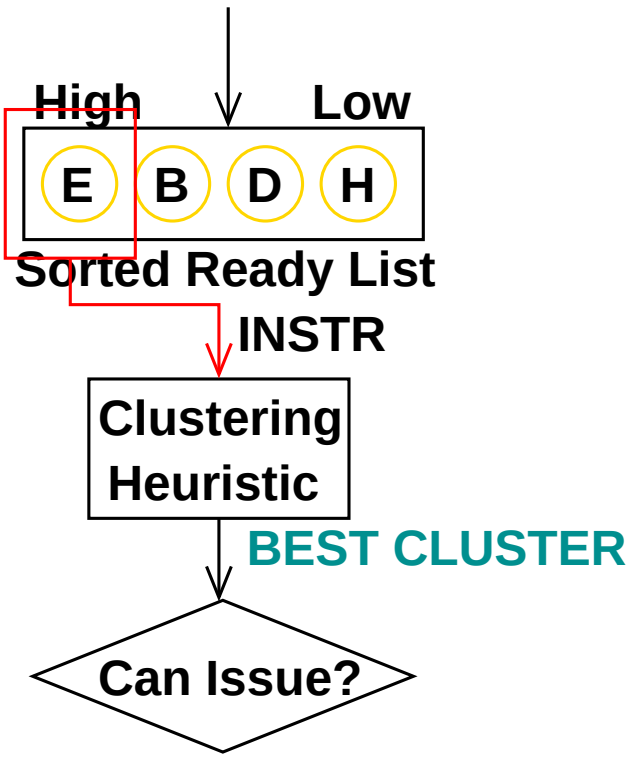


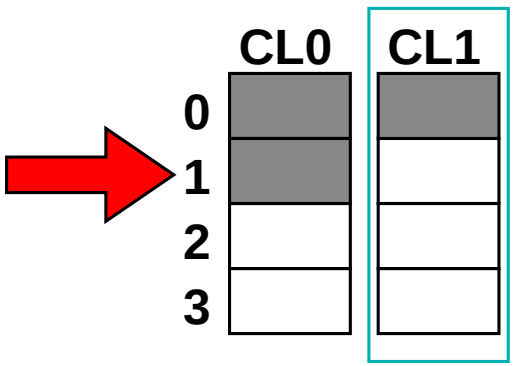
LUCAS Scheduler



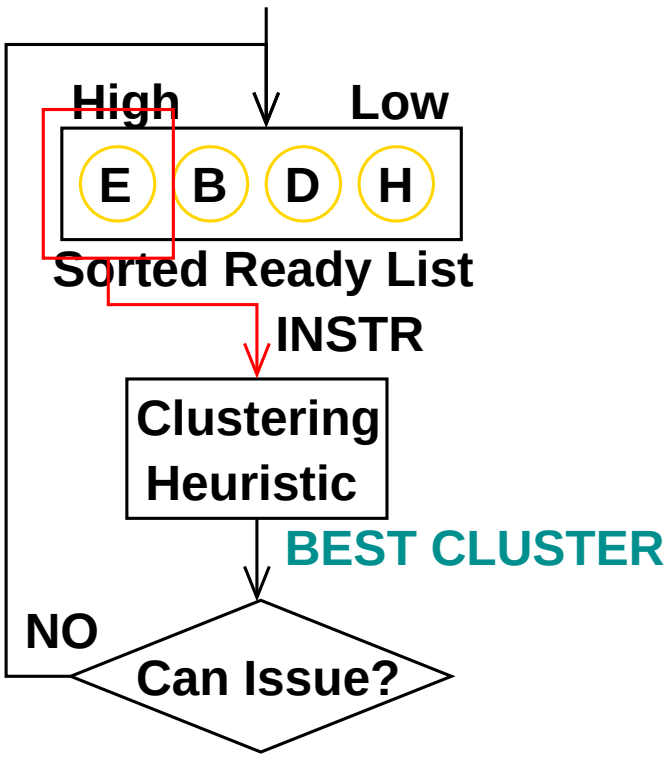


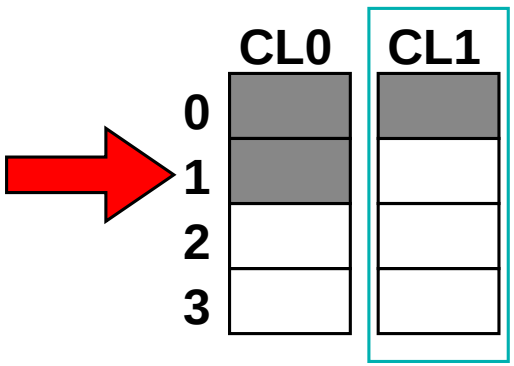
LUCAS Scheduler



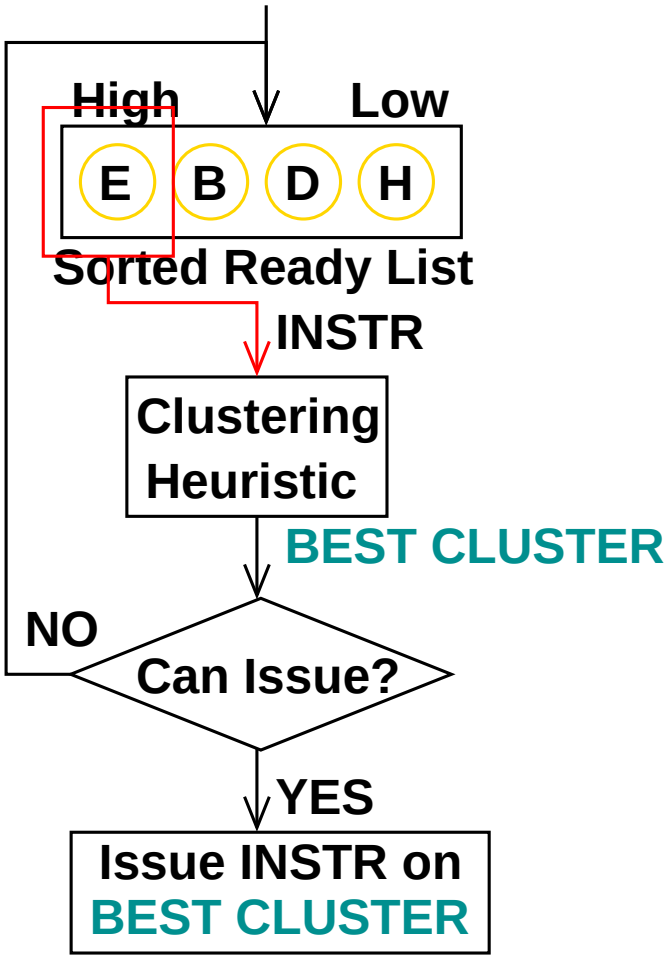


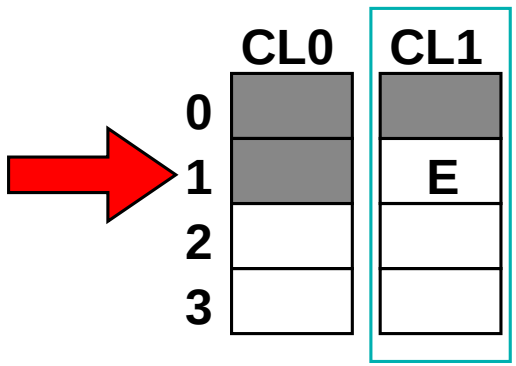
LUCAS Scheduler



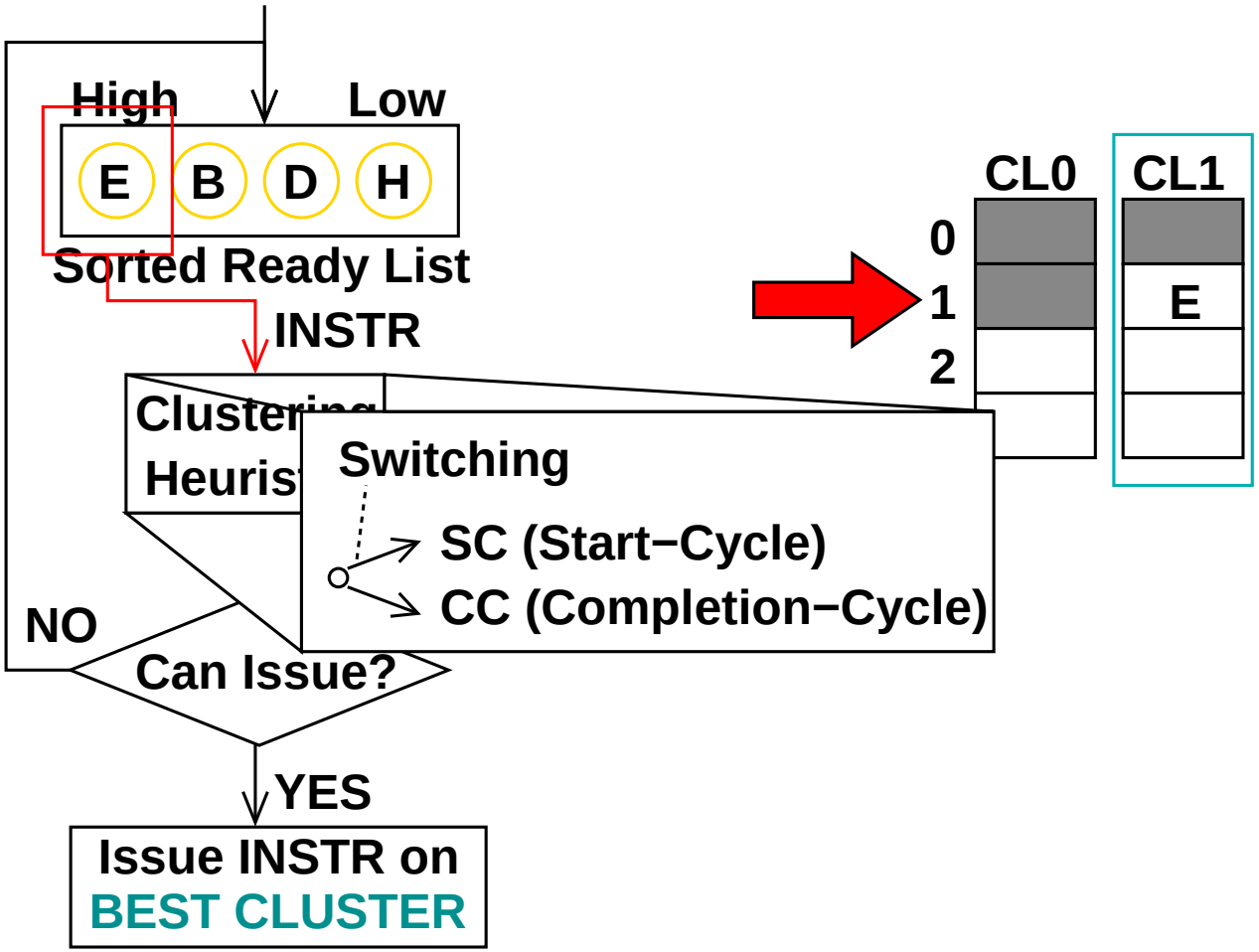


LUCAS Scheduler



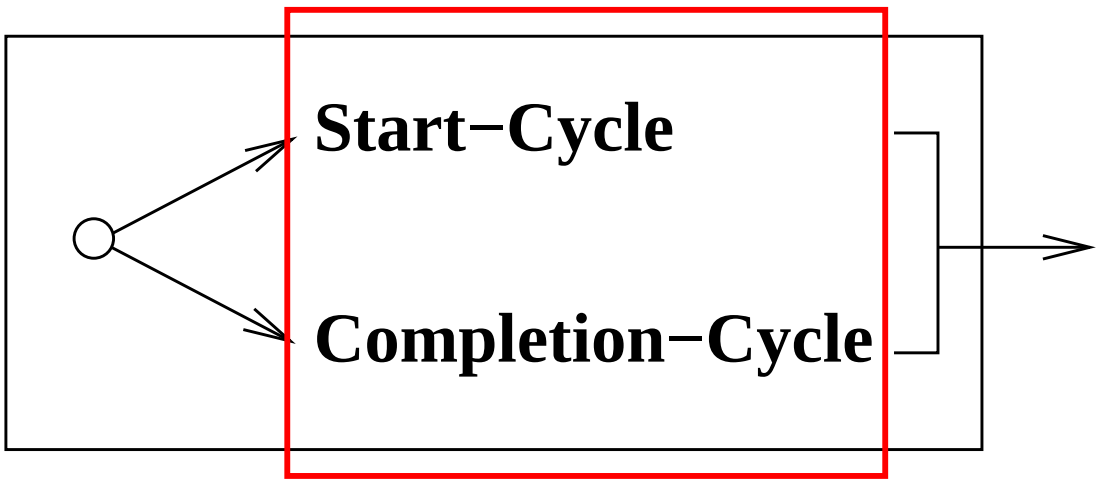


LUCAS Scheduler

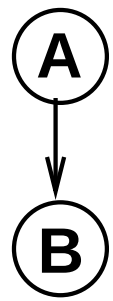


- Switch between aggressive SC and conservative CC at an instruction-level granularity

Clustering heuristics



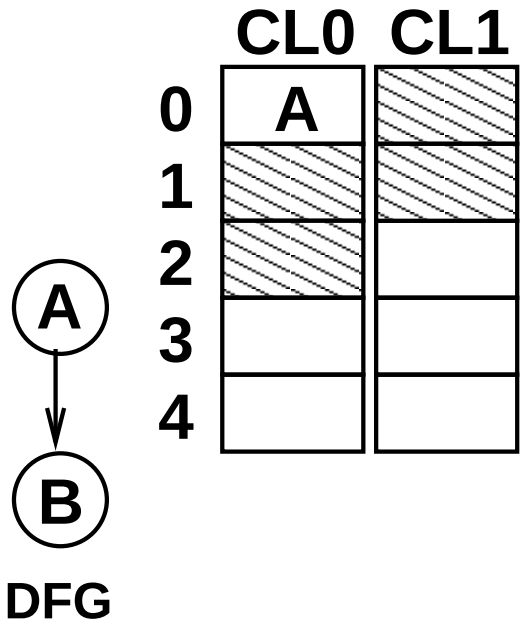
Clustering heuristics



DFG

Clustering heuristics

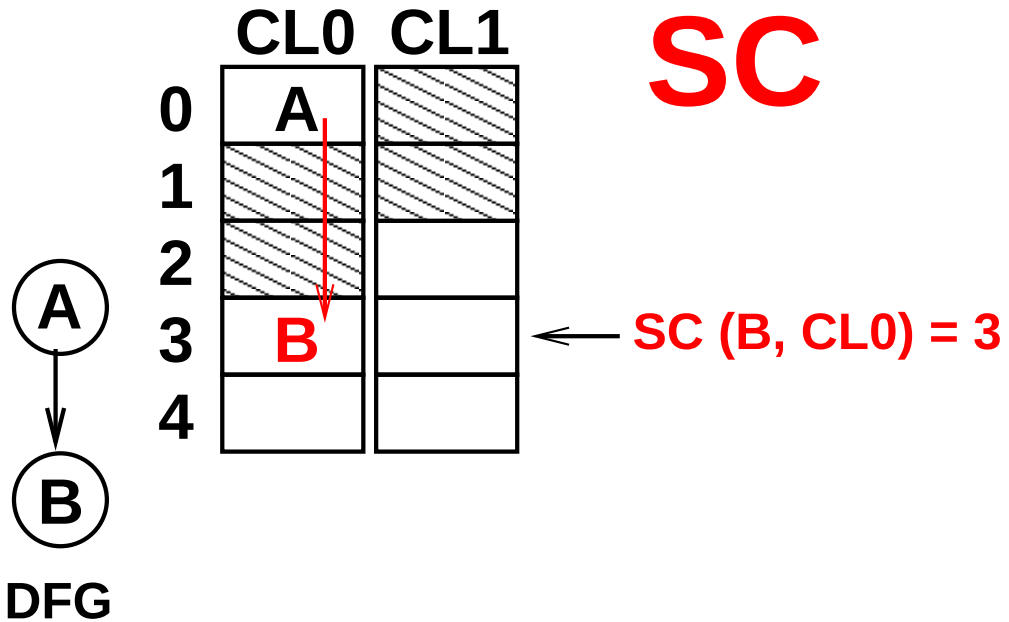
- Start-Cycle (SC) is the earliest cycle an instr. can be scheduled at a given cluster. One-way inter-cluster latency.



SC

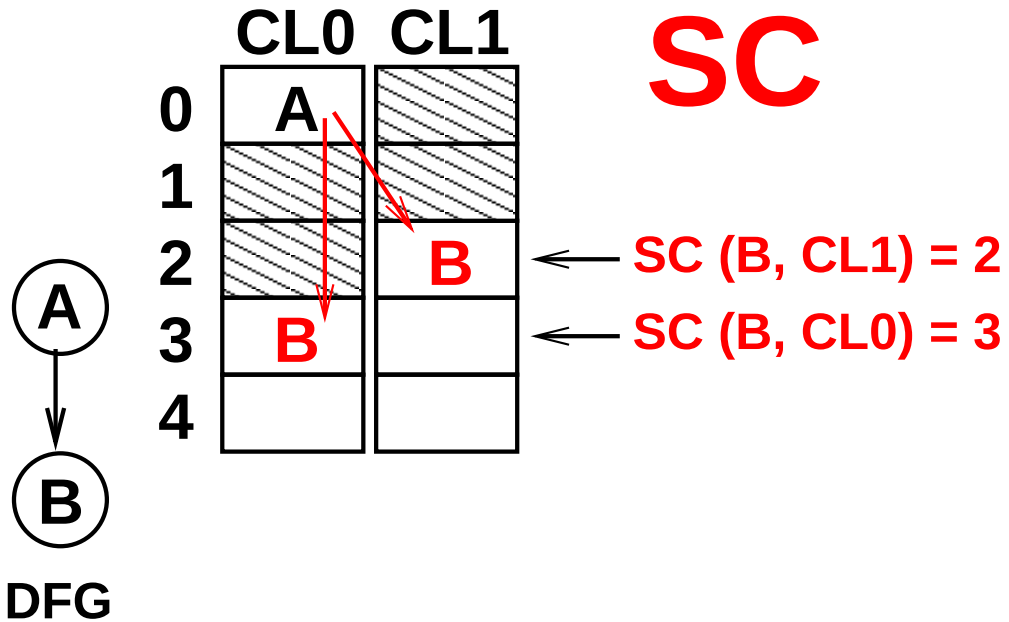
Clustering heuristics

- Start-Cycle (SC) is the earliest cycle an instr. can be scheduled at a given cluster. One-way inter-cluster latency.



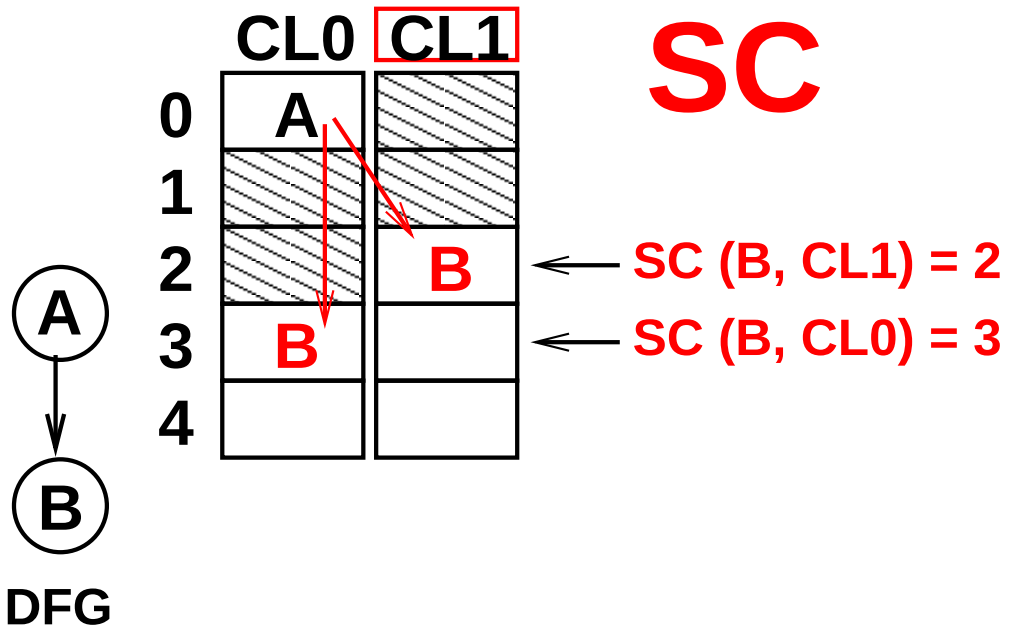
Clustering heuristics

- Start-Cycle (SC) is the earliest cycle an instr. can be scheduled at a given cluster. One-way inter-cluster latency.



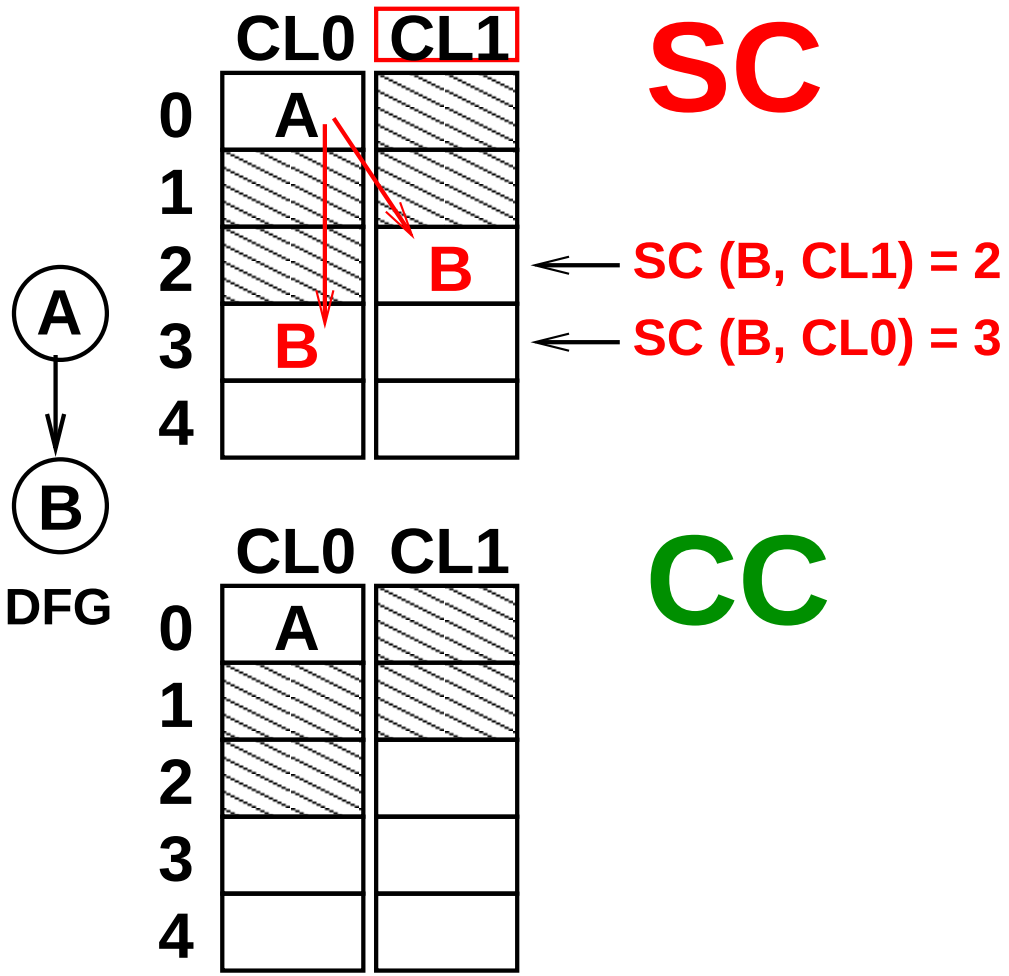
Clustering heuristics

- Start-Cycle (SC) is the earliest cycle an instr. can be scheduled at a given cluster. One-way inter-cluster latency.



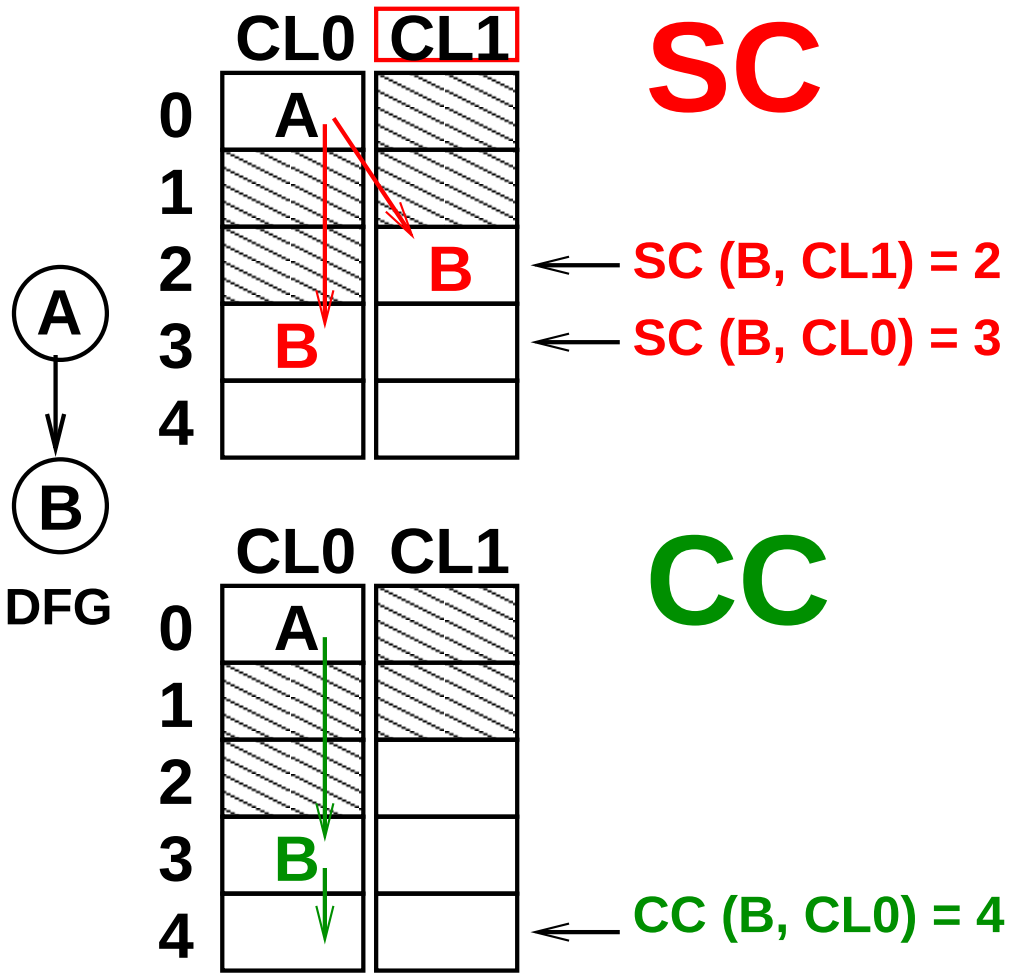
Clustering heuristics

- Start-Cycle (SC) is the earliest cycle an instr. can be scheduled at a given cluster. One-way inter-cluster latency.
- Completion-Cycle (CC) is the earliest cycle that an instr. can move its data to its successors. Round-trip inter-cluster latency.



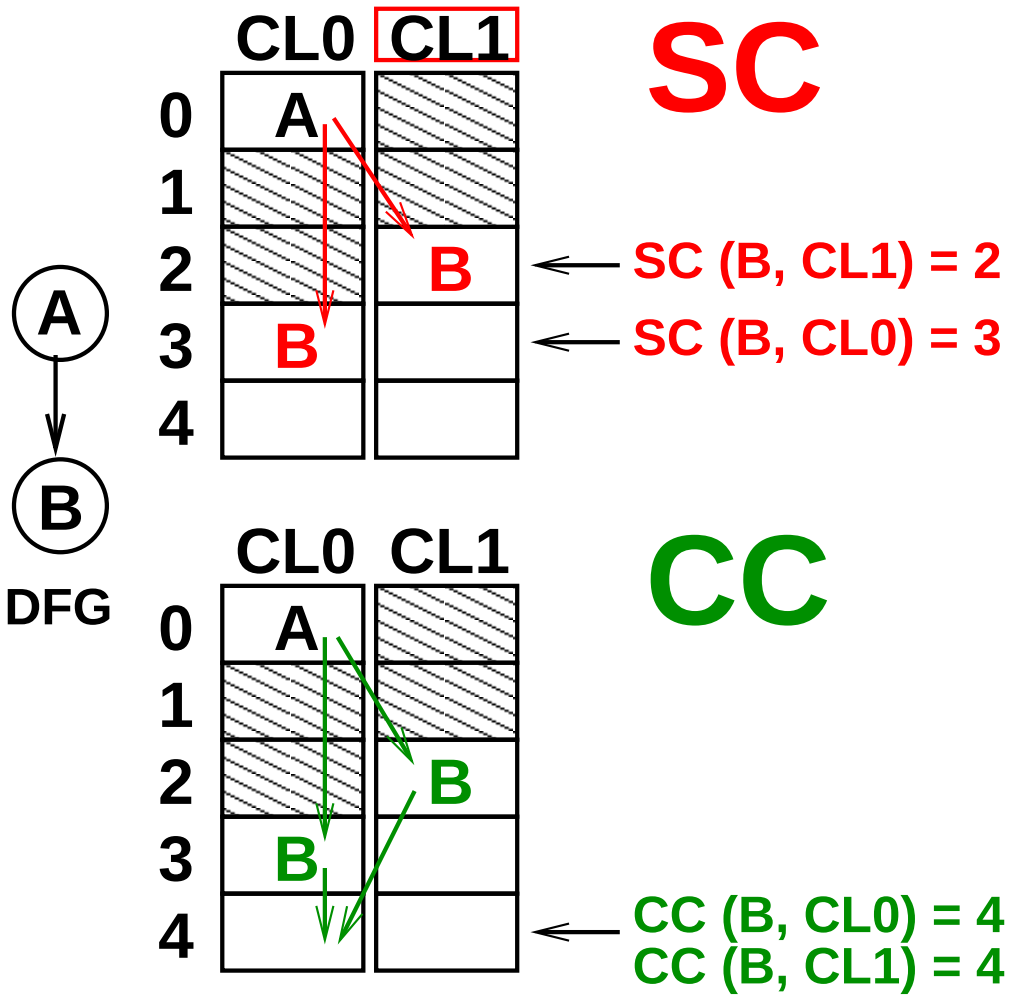
Clustering heuristics

- Start-Cycle (SC) is the earliest cycle an instr. can be scheduled at a given cluster. One-way inter-cluster latency.
- Completion-Cycle (CC) is the earliest cycle that an instr. can move its data to its successors. Round-trip inter-cluster latency.



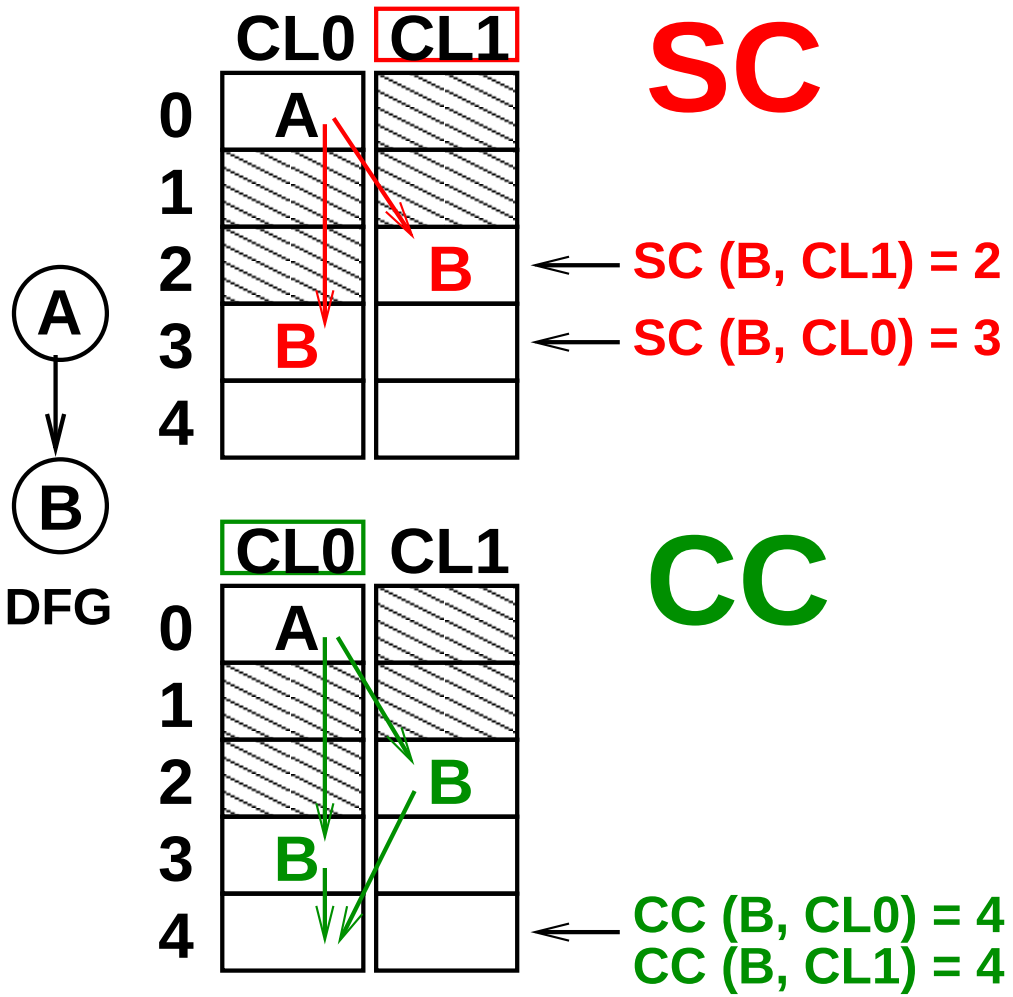
Clustering heuristics

- Start-Cycle (SC) is the earliest cycle an instr. can be scheduled at a given cluster. One-way inter-cluster latency.
- Completion-Cycle (CC) is the earliest cycle that an instr. can move its data to its successors. Round-trip inter-cluster latency.

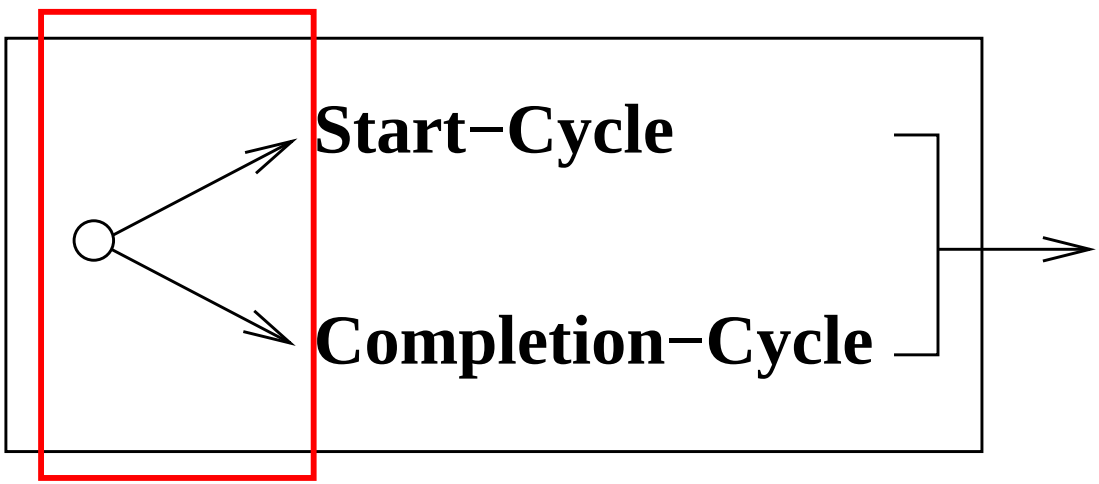


Clustering heuristics

- Start-Cycle (SC) is the earliest cycle an instr. can be scheduled at a given cluster. One-way inter-cluster latency.
- Completion-Cycle (CC) is the earliest cycle that an instr. can move its data to its successors. Round-trip inter-cluster latency.

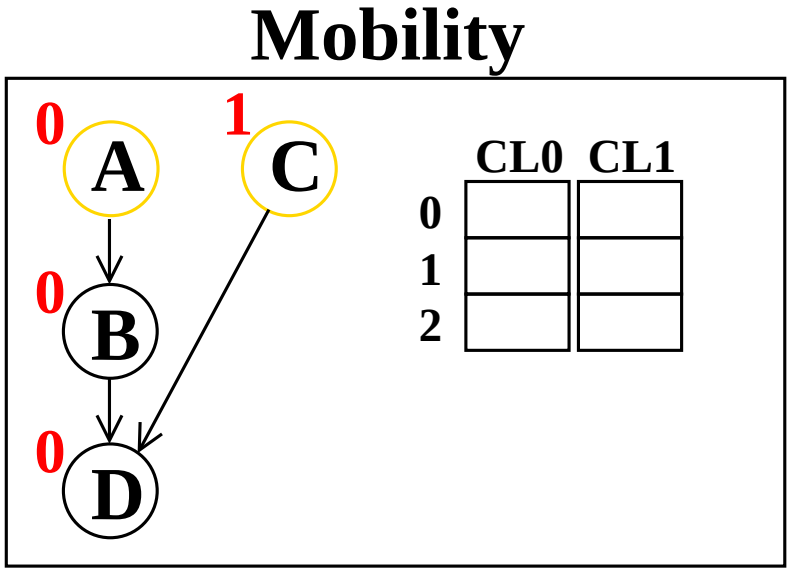


LUCAS switching heuristics



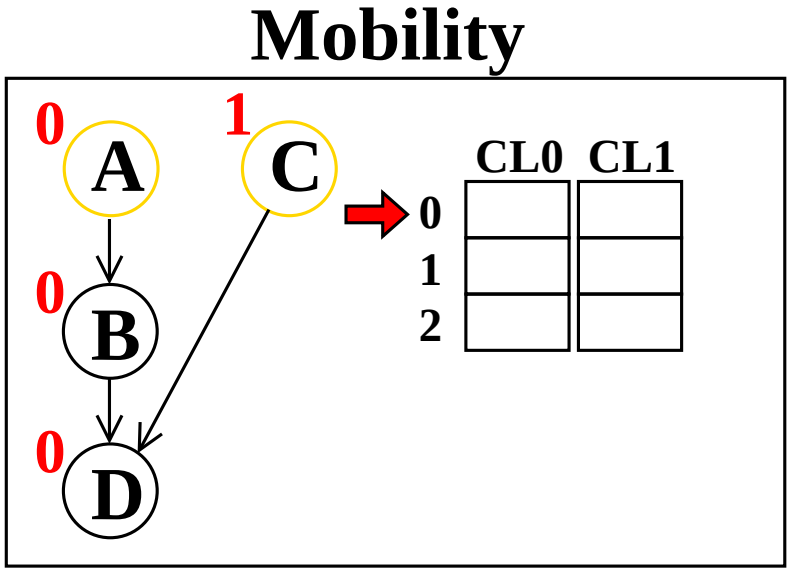
LUCAS switching heuristics - MOBILITY

- Mobility is a DFG-based heuristic.
- It measures the "slack" of an instruction in the DFG.
- If enough slack, cluster aggressively.



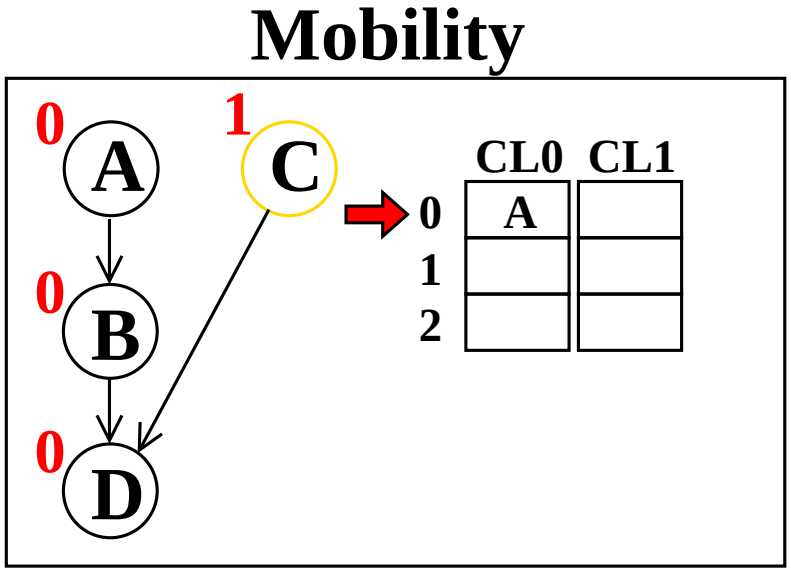
LUCAS switching heuristics - MOBILITY

- Mobility is a DFG-based heuristic.
- It measures the "slack" of an instruction in the DFG.
- If enough slack, cluster aggressively.



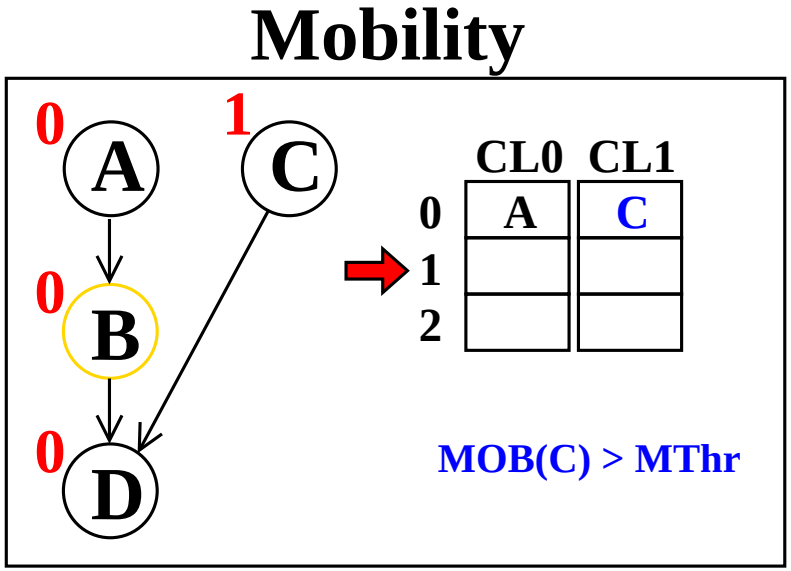
LUCAS switching heuristics - MOBILITY

- Mobility is a DFG-based heuristic.
- It measures the "slack" of an instruction in the DFG.
- If enough slack, cluster aggressively.
- Instructions normally clustered with CC heuristic.



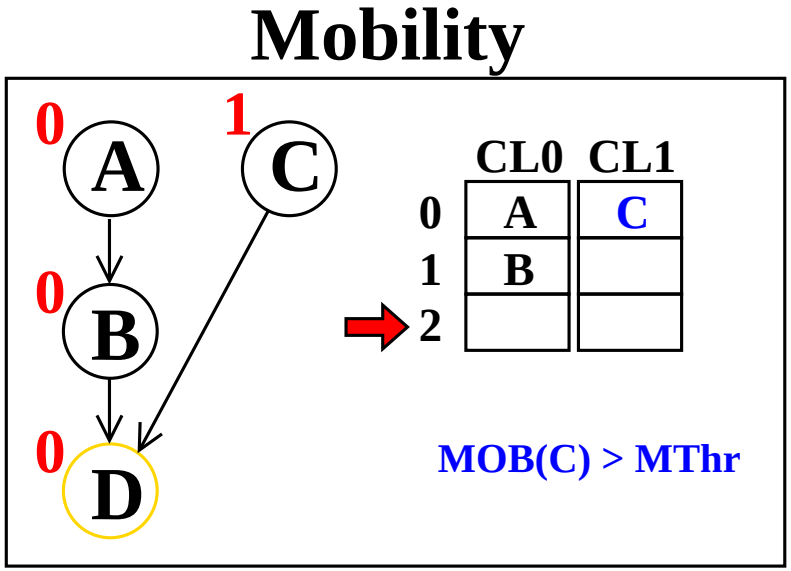
LUCAS switching heuristics - MOBILITY

- Mobility is a DFG-based heuristic.
- It measures the "slack" of an instruction in the DFG.
- If enough slack, cluster aggressively.
- Instructions normally clustered with CC heuristic.



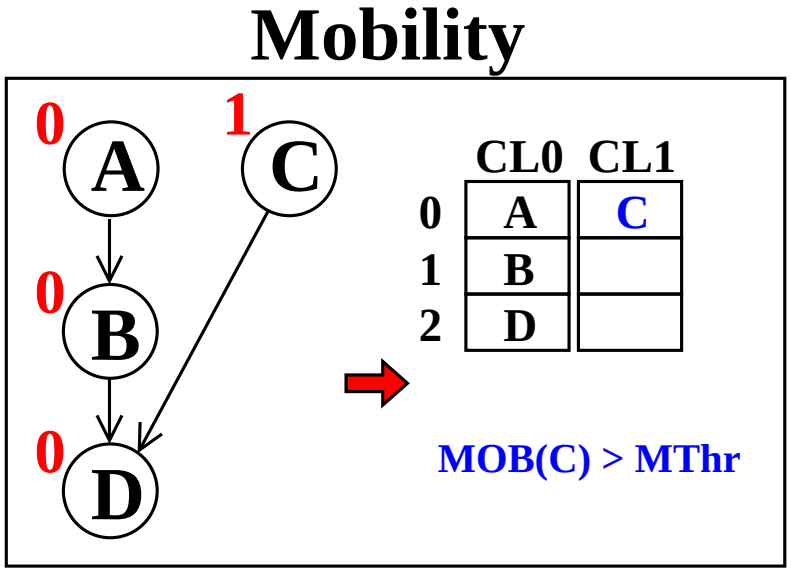
LUCAS switching heuristics - MOBILITY

- Mobility is a DFG-based heuristic.
- It measures the "slack" of an instruction in the DFG.
- If enough slack, cluster aggressively.
- Instructions normally clustered with CC heuristic.
- High Mobility dictates using SC heuristic.



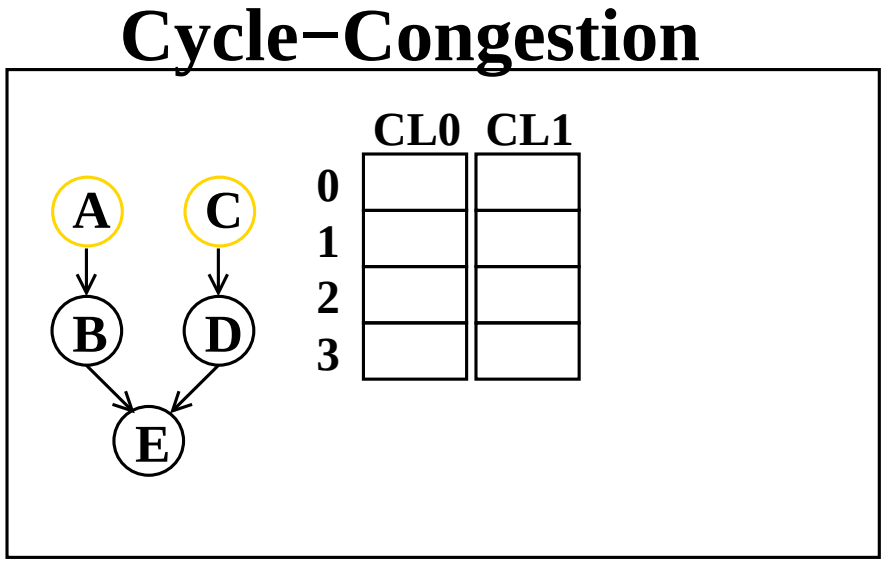
LUCAS switching heuristics - MOBILITY

- Mobility is a DFG-based heuristic.
- It measures the "slack" of an instruction in the DFG.
- If enough slack, cluster aggressively.
- Instructions normally clustered with CC heuristic.
- High Mobility dictates using SC heuristic.



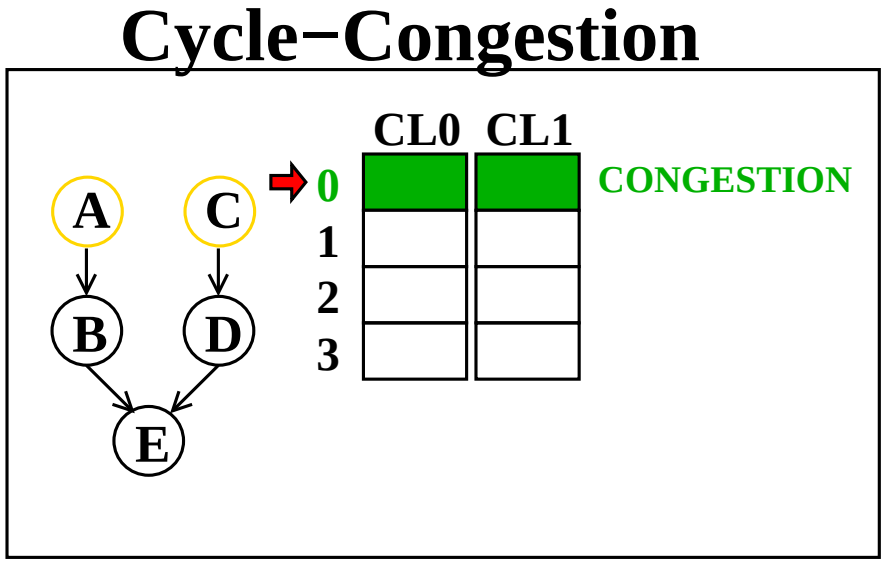
LUCAS switching heuristics - CONGESTION

- Congestion is a resource-based heuristic
- It measures how "full" the resources are at current scheduling cycle.
- If resources full, cluster aggressively.



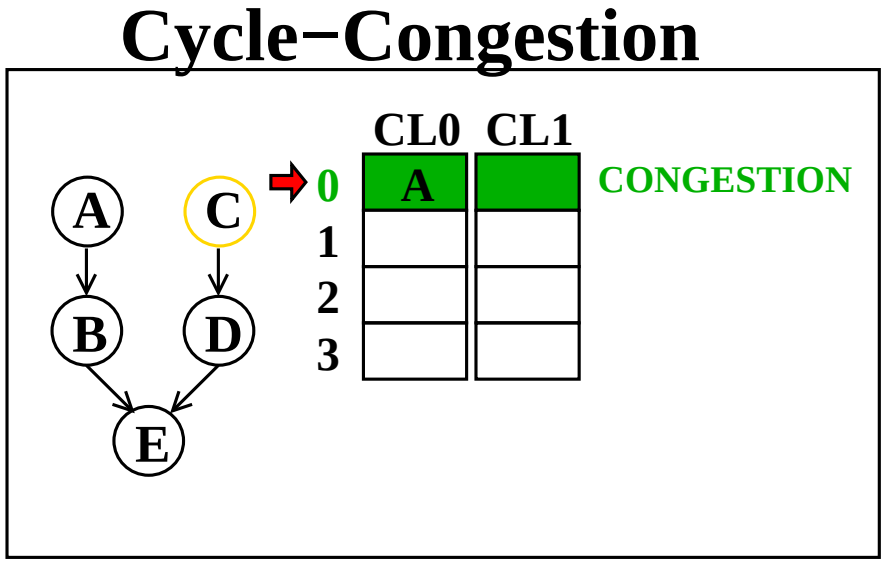
LUCAS switching heuristics - CONGESTION

- Congestion is a resource-based heuristic
- It measures how "full" the resources are at current scheduling cycle.
- If resources full, cluster aggressively.
- Instructions normally clustered with CC heuristic.



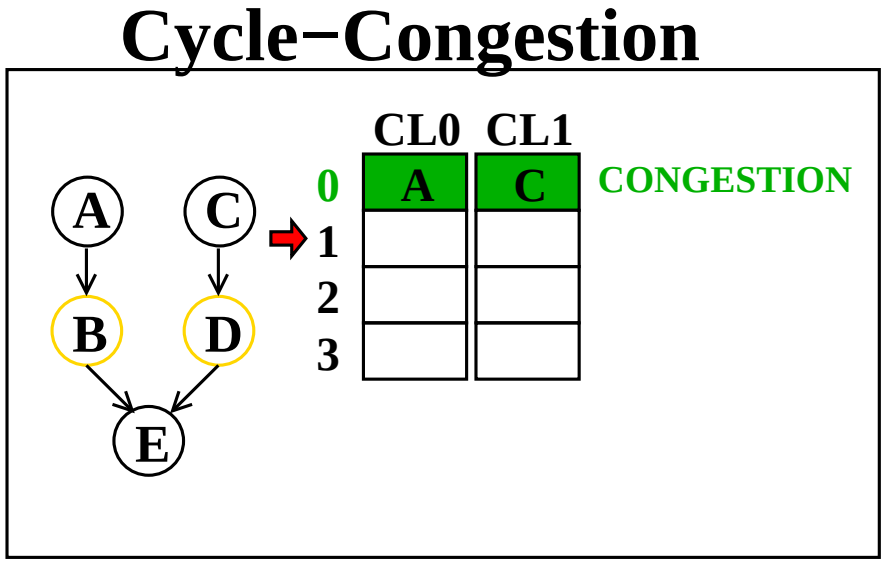
LUCAS switching heuristics - CONGESTION

- Congestion is a resource-based heuristic
- It measures how "full" the resources are at current scheduling cycle.
- If resources full, cluster aggressively.
- Instructions normally clustered with CC heuristic.
- High Congestion: use SC heuristic.



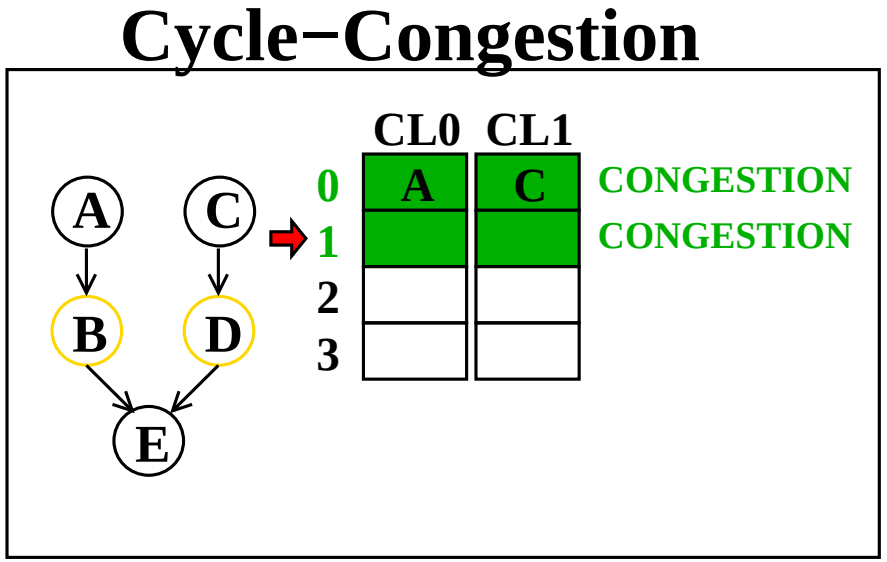
LUCAS switching heuristics - CONGESTION

- Congestion is a resource-based heuristic
- It measures how "full" the resources are at current scheduling cycle.
- If resources full, cluster aggressively.
- Instructions normally clustered with CC heuristic.
- High Congestion: use SC heuristic.



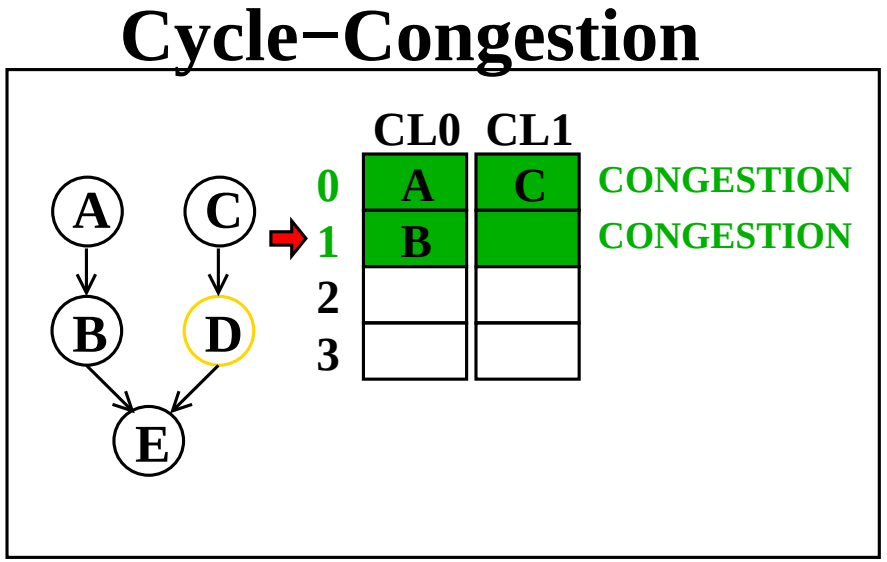
LUCAS switching heuristics - CONGESTION

- Congestion is a resource-based heuristic
- It measures how "full" the resources are at current scheduling cycle.
- If resources full, cluster aggressively.
- Instructions normally clustered with CC heuristic.
- High Congestion: use SC heuristic.



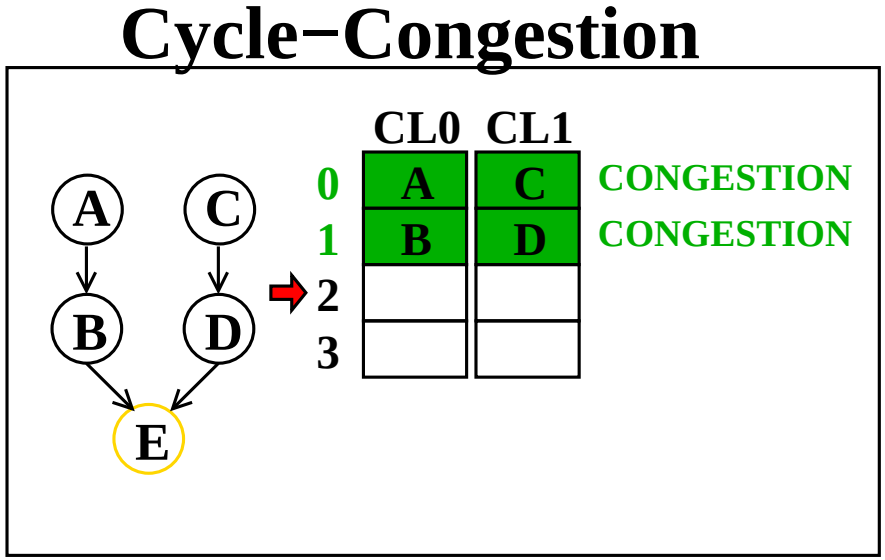
LUCAS switching heuristics - CONGESTION

- Congestion is a resource-based heuristic
- It measures how "full" the resources are at current scheduling cycle.
- If resources full, cluster aggressively.
- Instructions normally clustered with CC heuristic.
- High Congestion: use SC heuristic.



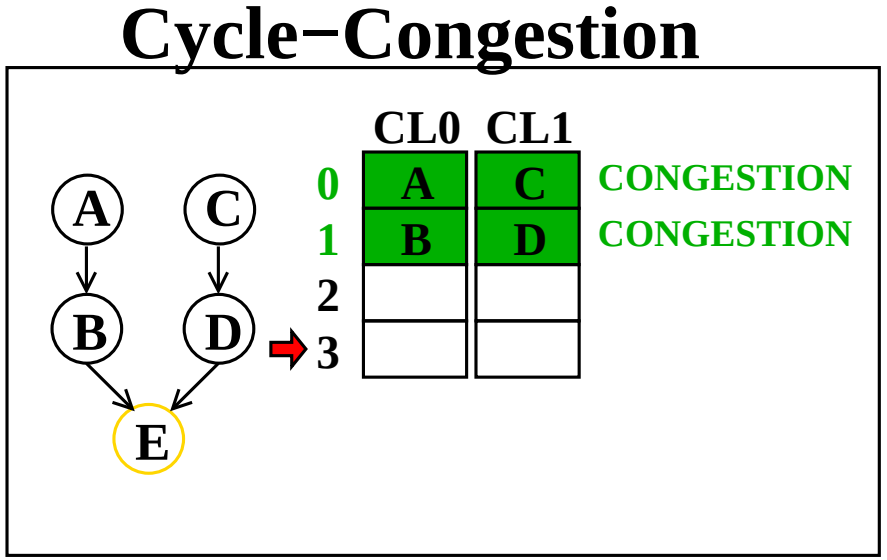
LUCAS switching heuristics - CONGESTION

- Congestion is a resource-based heuristic
- It measures how "full" the resources are at current scheduling cycle.
- If resources full, cluster aggressively.
- Instructions normally clustered with CC heuristic.
- High Congestion: use SC heuristic.



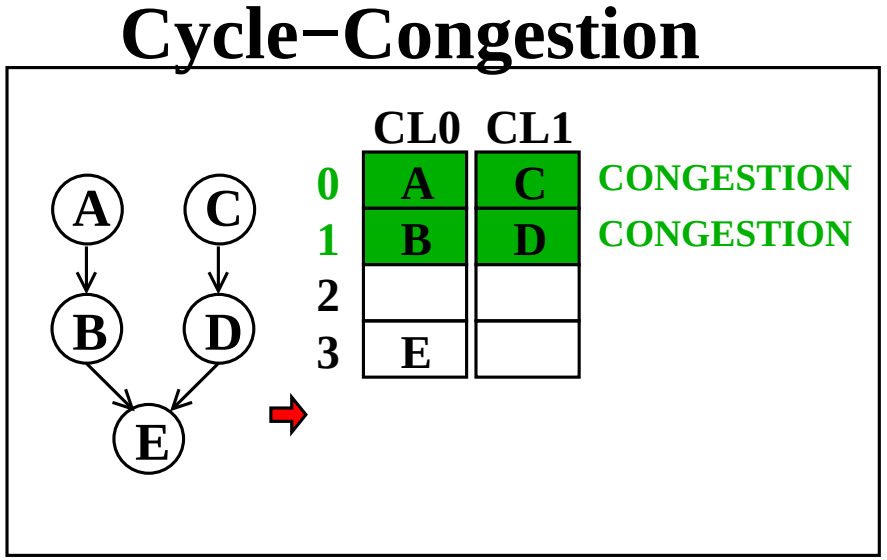
LUCAS switching heuristics - CONGESTION

- Congestion is a resource-based heuristic
- It measures how "full" the resources are at current scheduling cycle.
- If resources full, cluster aggressively.
- Instructions normally clustered with CC heuristic.
- High Congestion: use SC heuristic.

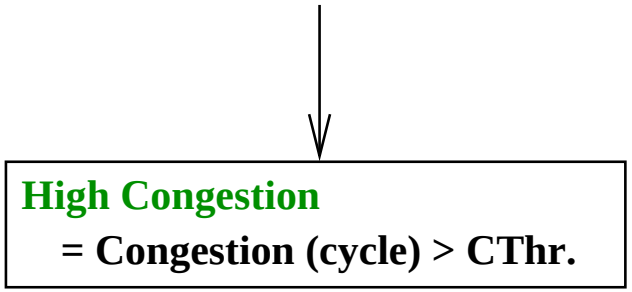


LUCAS switching heuristics - CONGESTION

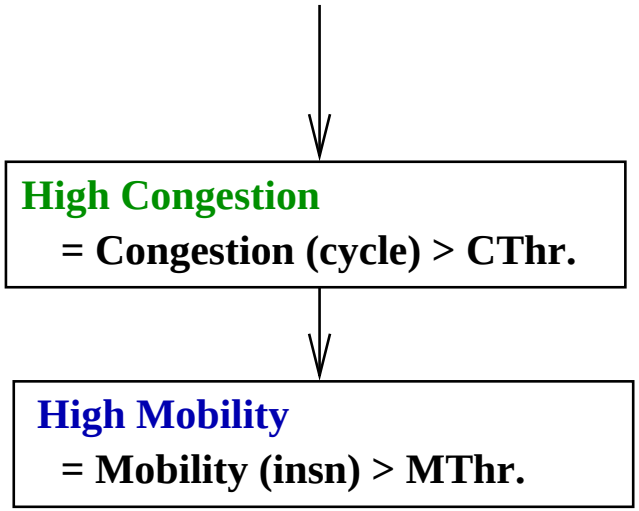
- Congestion is a resource-based heuristic
- It measures how "full" the resources are at current scheduling cycle.
- If resources full, cluster aggressively.
- Instructions normally clustered with CC heuristic.
- High Congestion: use SC heuristic.



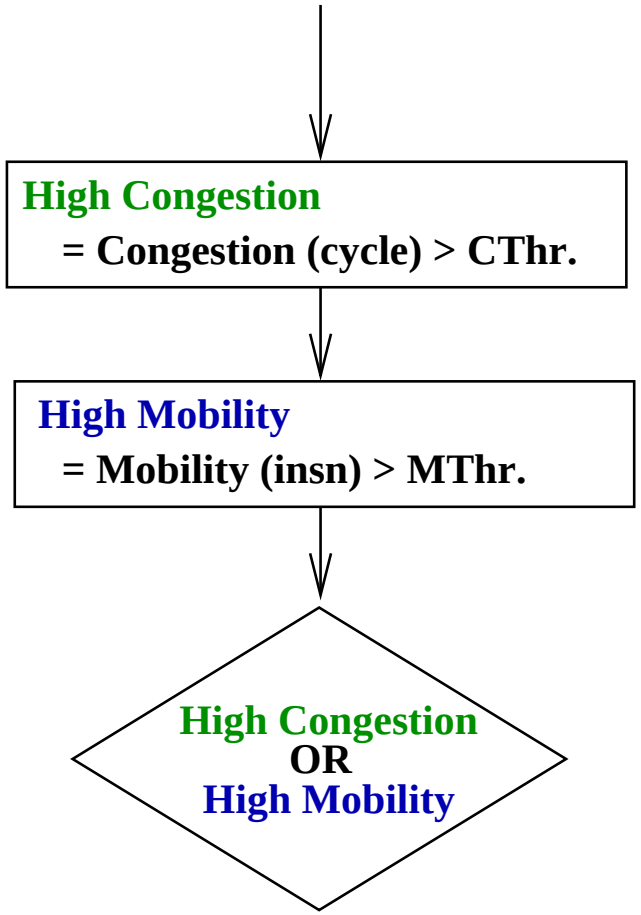
LUCAS Heuristic



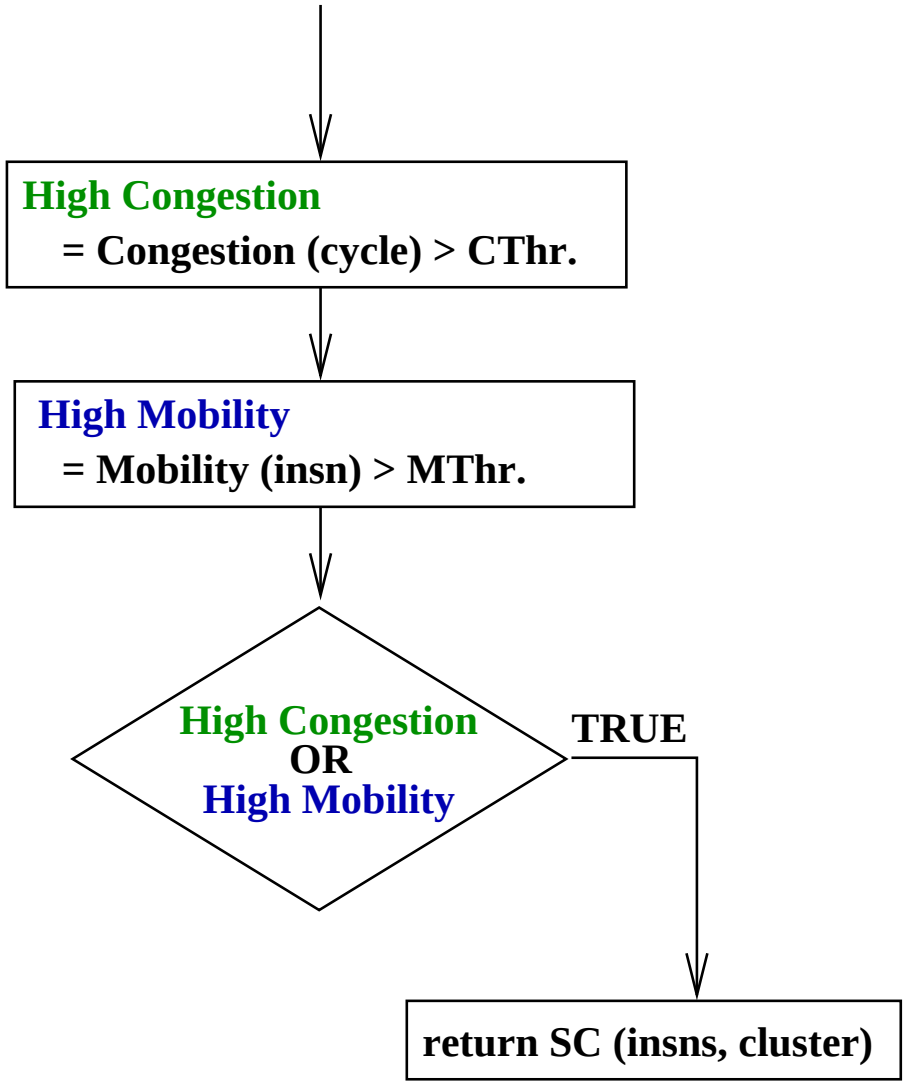
LUCAS Heuristic



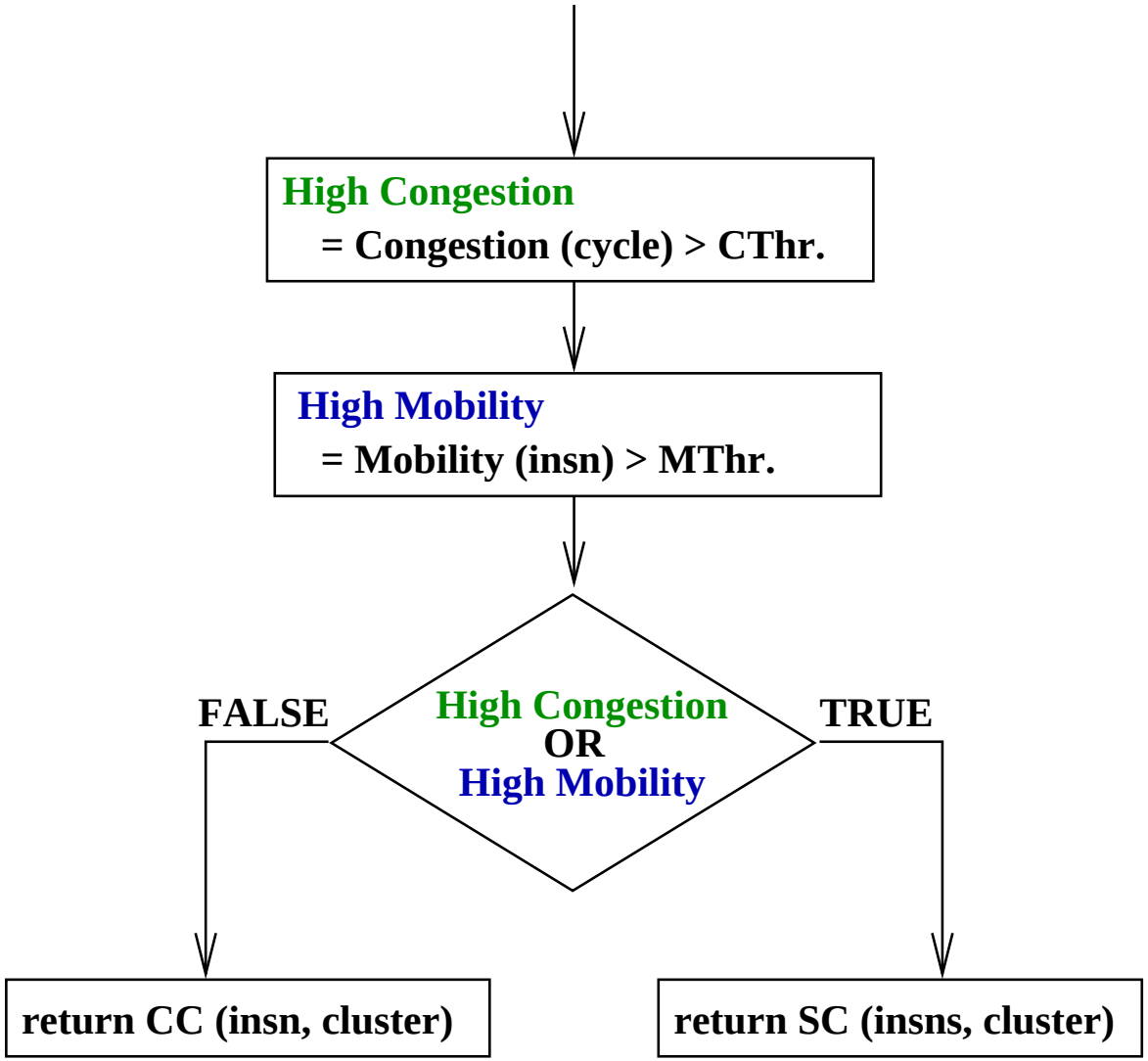
LUCAS Heuristic



LUCAS Heuristic



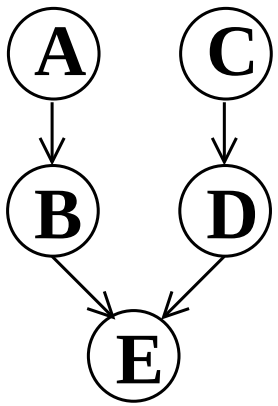
LUCAS Heuristic



Outline

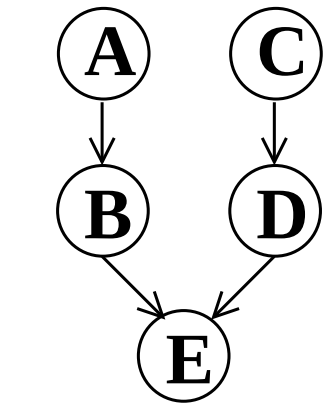
LUCAS Congestion Example

1 CYCLE **2 CYCLES** **3 CYCLES**



Data Flow Graph

LUCAS Congestion Example

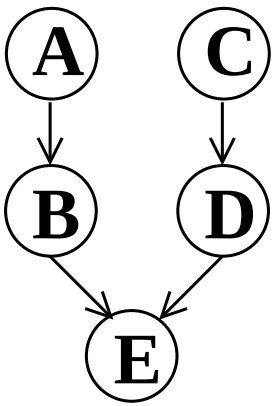


Data Flow Graph

SC

1 CYCLE 2 CYCLES 3 CYCLES

LUCAS Congestion Example



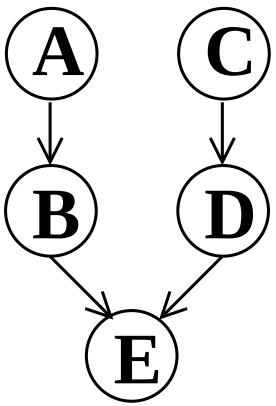
Data Flow Graph

1 CYCLE 2 CYCLES 3 CYCLES

SC

	CL0	CL1
0	A	C
1	B	D
2		
3	E	

LUCAS Congestion Example

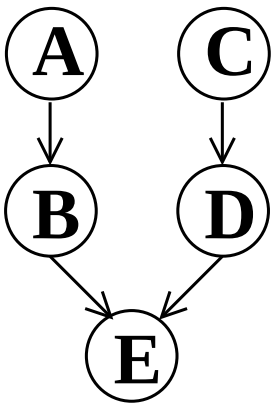


Data Flow Graph

SC

	1 CYCLE		2 CYCLES		3 CYCLES
	CL0	CL1	CL0	CL1	
0	A	C	A	C	
1	B	D	B	D	
2					
3	E				
			E		

LUCAS Congestion Example

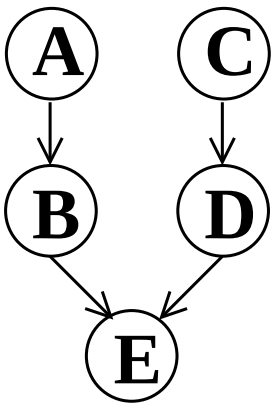


Data Flow Graph

SC

	1 CYCLE		2 CYCLES		3 CYCLES	
	CL0	CL1	CL0	CL1	CL0	CL1
0	A	C	A	C	A	C
1	B	D	B	D	B	D
2						
3	E					
			E		E	

LUCAS Congestion Example



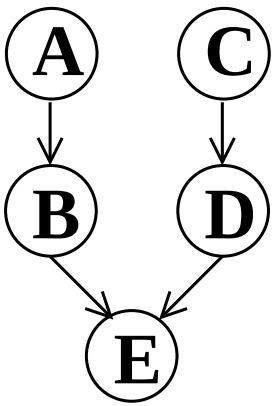
Data Flow Graph

SC

	1 CYCLE		2 CYCLES		3 CYCLES	
	CL0	CL1	CL0	CL1	CL0	CL1
0	A	C	A	C	A	C
1	B	D	B	D	B	D
2						
3	E					
			E			
					E	

CC

LUCAS Congestion Example



Data Flow Graph

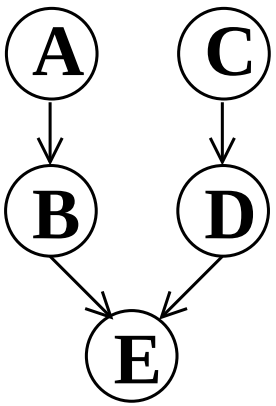
SC

	1 CYCLE CL0	1 CYCLE CL1	2 CYCLES CL0	2 CYCLES CL1	3 CYCLES CL0	3 CYCLES CL1
0	A	C	A	C	A	C
1	B	D	B	D	B	D
2						
3	E					
			E		E	

CC

0	A
1	C
2	B
3	D
4	E

LUCAS Congestion Example



Data Flow Graph

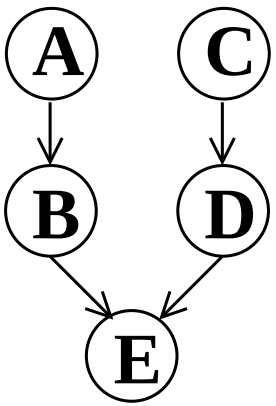
SC

	1 CYCLE CL0	1 CYCLE CL1	2 CYCLES CL0	2 CYCLES CL1	3 CYCLES CL0	3 CYCLES CL1
0	A	C	A	C	A	C
1	B	D	B	D	B	D
2						
3	E					
			E		E	

CC

	1 CYCLE		2 CYCLES	
0	A		A	
1	C		C	
2	B		B	
3	D		D	
4	E		E	

LUCAS Congestion Example



Data Flow Graph

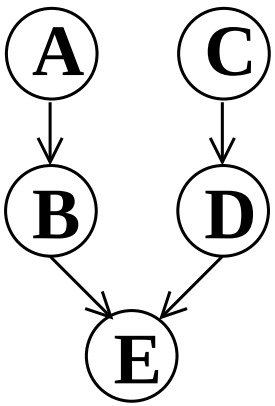
SC

	1 CYCLE		2 CYCLES		3 CYCLES	
	CL0	CL1	CL0	CL1	CL0	CL1
0	A	C	A	C	A	C
1	B	D	B	D	B	D
2						
3	E					
			E			
					E	

CC

0	A		A		A	
1	C		C		C	
2	B		B		B	
3	D		D		D	
4	E		E		E	

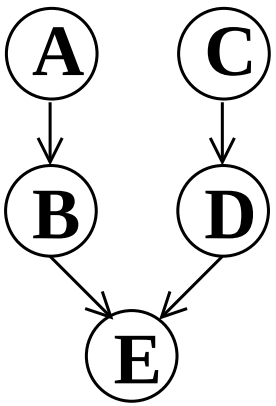
LUCAS Congestion Example



Data Flow Graph

	1 CYCLE		2 CYCLES		3 CYCLES	
	CL0	CL1	CL0	CL1	CL0	CL1
SC						
0	A	C	A	C	A	C
1	B	D	B	D	B	D
2						
3	E					
			E			
					E	
CC						
0	A		A		A	
1	C		C		C	
2	B		B		B	
3	D		D		D	
4	E		E		E	
LUCAS						

LUCAS Congestion Example

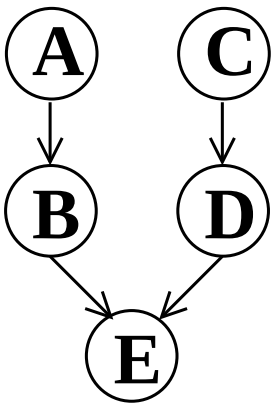


Data Flow Graph

	1 CYCLE		2 CYCLES		3 CYCLES	
	CL0	CL1	CL0	CL1	CL0	CL1
SC						
0	A	C	A	C	A	C
1	B	D	B	D	B	D
2						
3	E					
			E			
					E	
CC						
0	A		A		A	
1	C		C		C	
2	B		B		B	
3	D		D		D	
4	E		E		E	
LUCAS						
0	A	C				
1	B	D				
2						
3	E					

CONGESTION

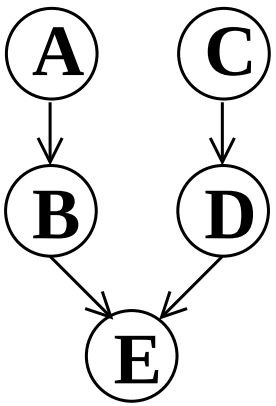
LUCAS Congestion Example



Data Flow Graph

	1 CYCLE		2 CYCLES		3 CYCLES	
	CL0	CL1	CL0	CL1	CL0	CL1
SC						
0	A	C	A	C	A	C
1	B	D	B	D	B	D
2						
3	E					
			E			
					E	
CC						
0	A		A		A	
1	C		C		C	
2	B		B		B	
3	D		D		D	
4	E		E		E	
LUCAS						
0	A	C	A	C		
1	B	D	B	D		
2						
3	E					
	CONGESTION					
			E			

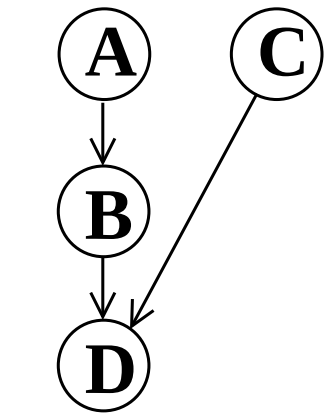
LUCAS Congestion Example



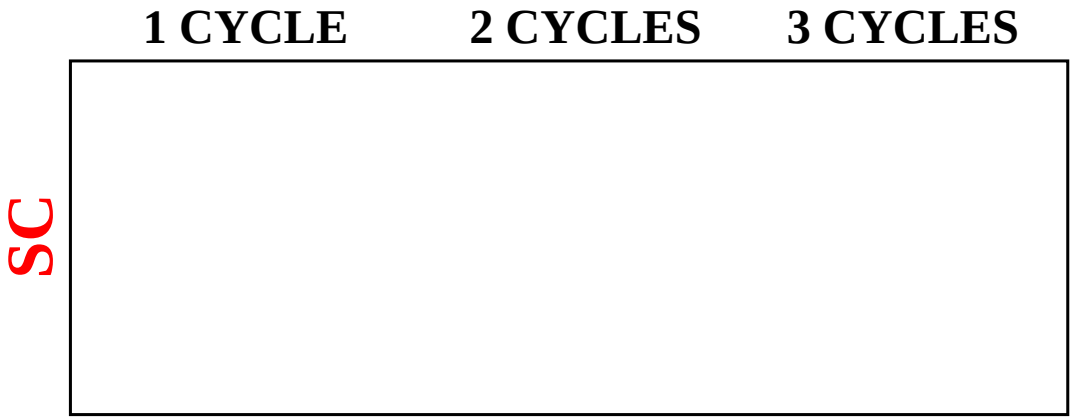
Data Flow Graph

	1 CYCLE		2 CYCLES		3 CYCLES	
	CL0	CL1	CL0	CL1	CL0	CL1
SC						
0	A	C	A	C	A	C
1	B	D	B	D	B	D
2						
3	E					
			E			
					E	
CC						
0	A		A		A	
1	C		C		C	
2	B		B		B	
3	D		D		D	
4	E		E		E	
LUCAS						
0	A	C	A	C	A	
1	B	D	B	D	C	
2					B	
3	E				D	
	CONGESTION		E		E	

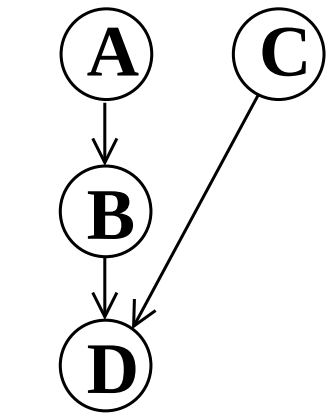
LUCAS Mobility Example



Data Flow Graph



LUCAS Mobility Example



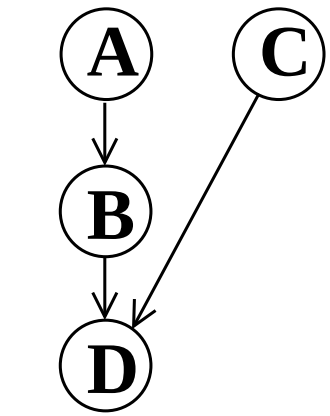
Data Flow Graph

1 CYCLE 2 CYCLES 3 CYCLES

SC

	CL0	CL1
0	A	C
1	B	
2	D	

LUCAS Mobility Example

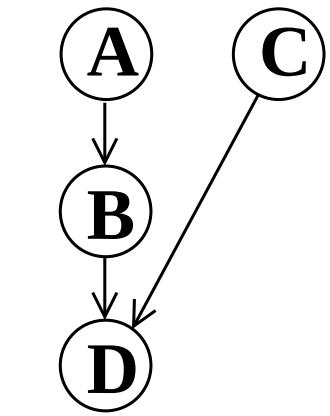


Data Flow Graph

SC

	1 CYCLE		2 CYCLES	
	CL0	CL1	CL0	CL1
0	A	C	A	C
1	B		B	
2	D			
			D	

LUCAS Mobility Example

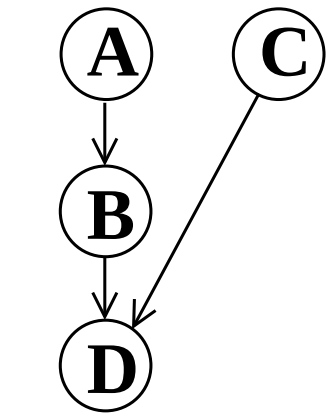


Data Flow Graph

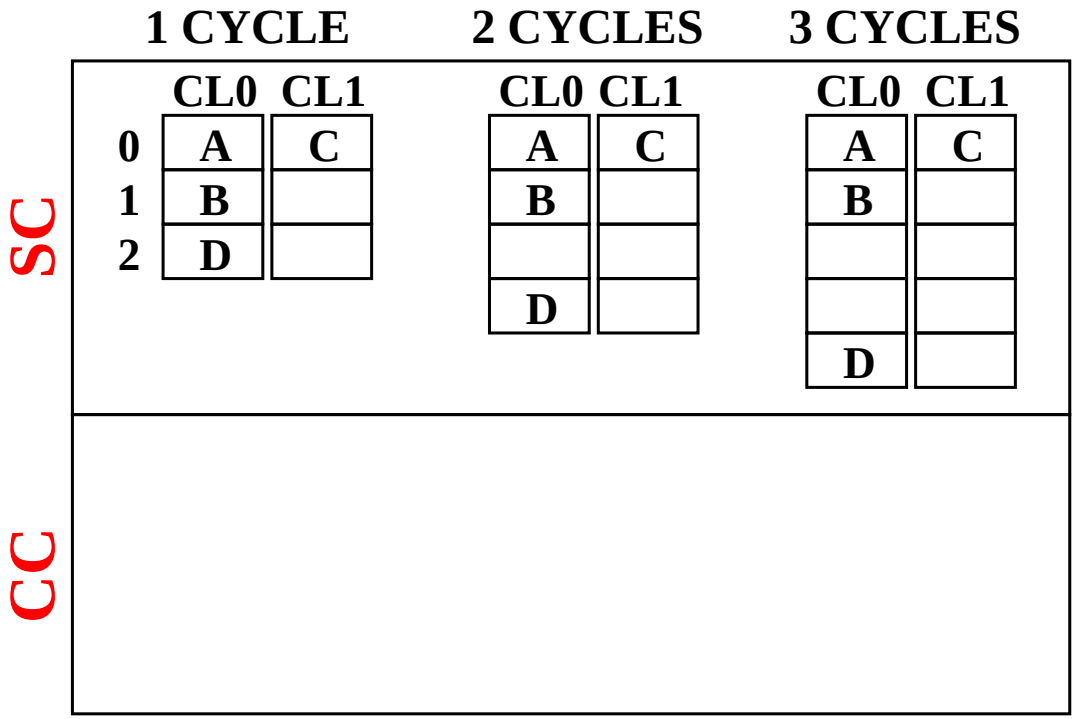
SC

	1 CYCLE		2 CYCLES		3 CYCLES	
	CL0	CL1	CL0	CL1	CL0	CL1
0	A	C	A	C	A	C
1	B		B		B	
2	D					
			D			
					D	

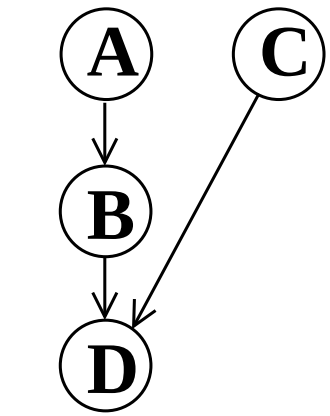
LUCAS Mobility Example



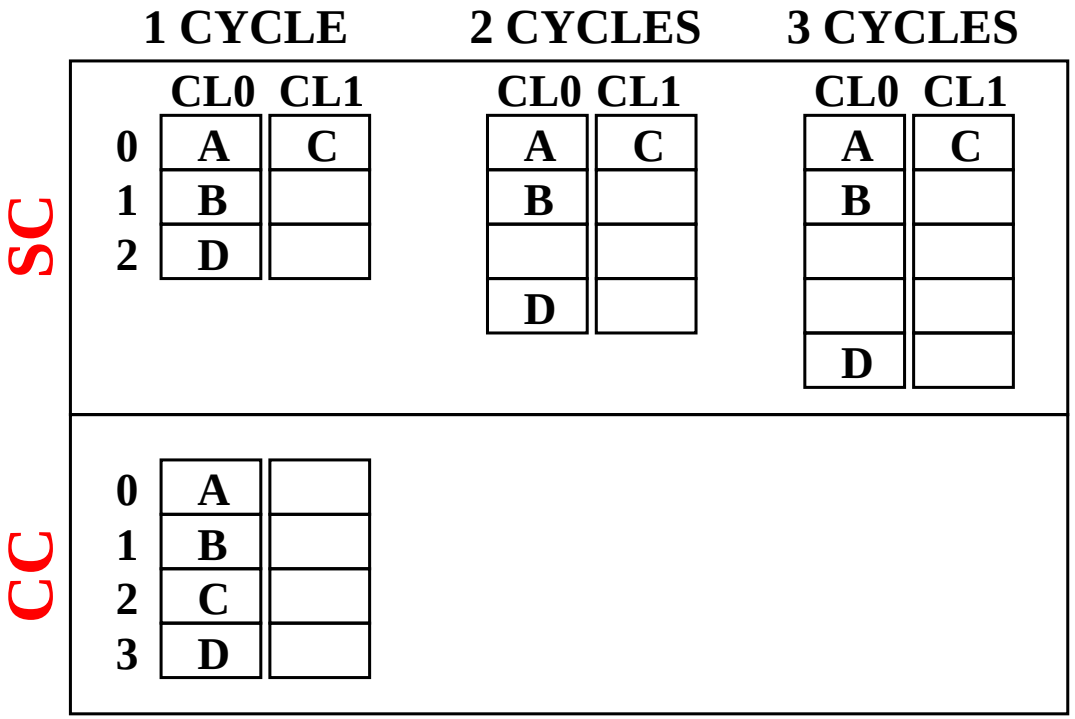
Data Flow Graph



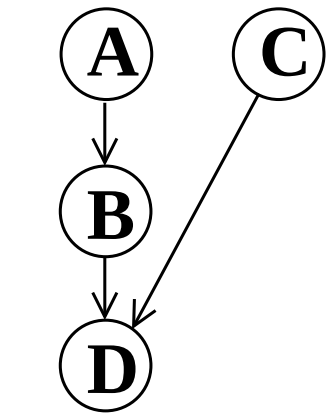
LUCAS Mobility Example



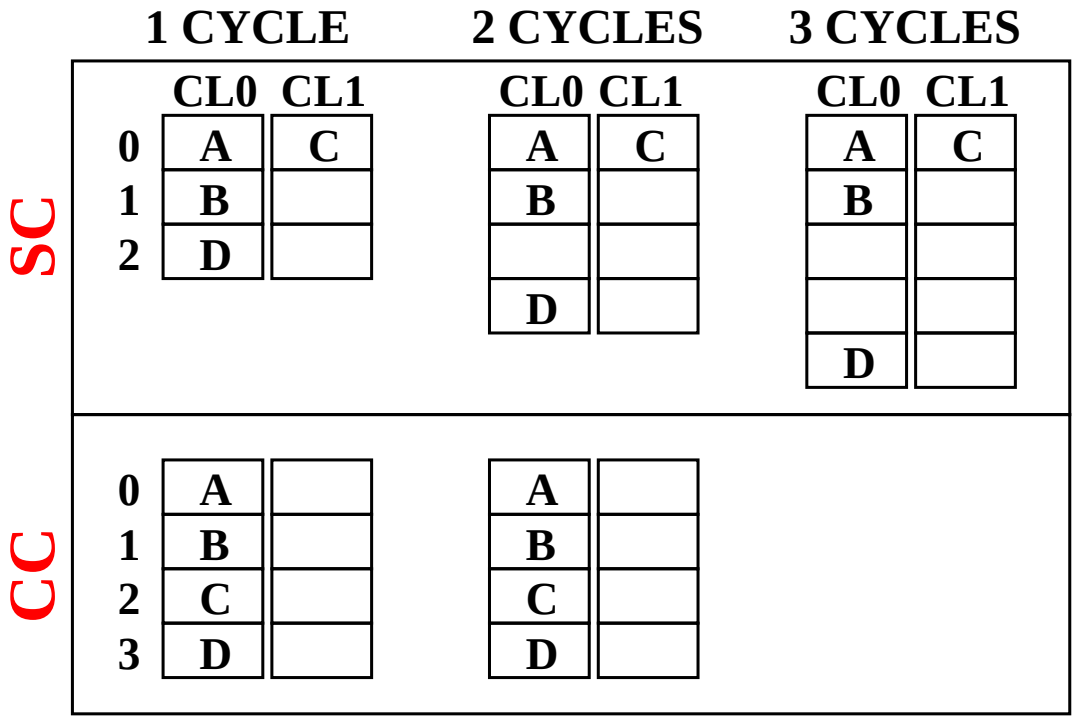
Data Flow Graph



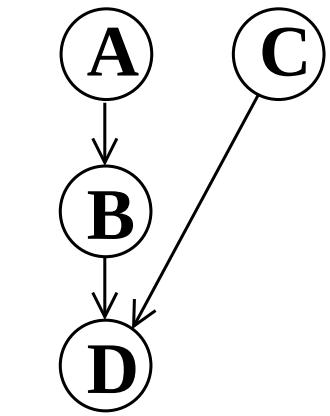
LUCAS Mobility Example



Data Flow Graph



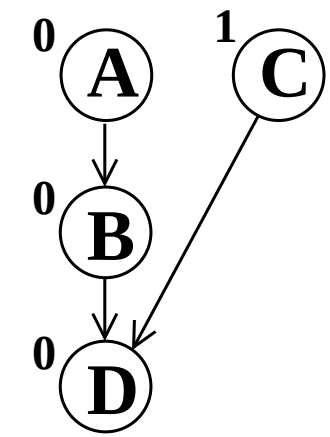
LUCAS Mobility Example



Data Flow Graph

	1 CYCLE		2 CYCLES		3 CYCLES	
SC	CL0	CL1	CL0	CL1	CL0	CL1
0	A	C	A	C	A	C
1	B		B		B	
2	D					
			D			
					D	
CC						
0	A		A		A	
1	B		B		B	
2	C		C		C	
3	D		D		D	

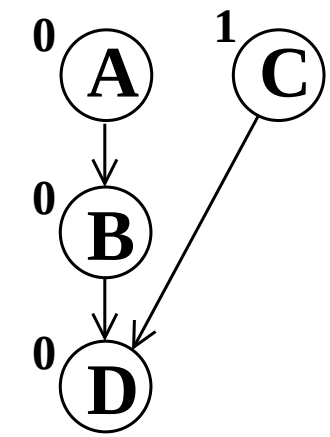
LUCAS Mobility Example



Data Flow Graph

	1 CYCLE		2 CYCLES		3 CYCLES	
	CL0	CL1	CL0	CL1	CL0	CL1
SC						
0	A	C	A	C	A	C
1	B		B		B	
2	D					
			D			
					D	
CC						
0	A		A		A	
1	B		B		B	
2	C		C		C	
3	D		D		D	
LUCAS						

LUCAS Mobility Example

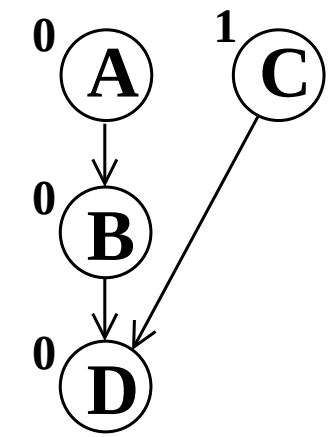


Data Flow Graph

		1 CYCLE		2 CYCLES		3 CYCLES	
		CL0	CL1	CL0	CL1	CL0	CL1
SC	0	A	C	A	C	A	C
	1	B		B		B	
	2	D					
				D			
						D	
CC	0	A		A		A	
	1	B		B		B	
	2	C		C		C	
	3	D		D		D	
LUCAS	0	A	C				
	1	B					
	2	D					

MOB(C) > MThr

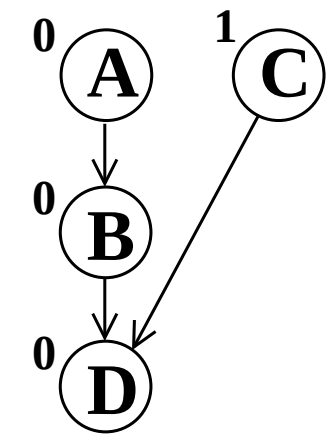
LUCAS Mobility Example



Data Flow Graph

		1 CYCLE		2 CYCLES		3 CYCLES	
		CL0	CL1	CL0	CL1	CL0	CL1
SC	0	A	C	A	C	A	C
	1	B		B		B	
	2	D					
				D			
						D	
CC	0	A		A		A	
	1	B		B		B	
	2	C		C		C	
	3	D		D		D	
LUCAS	0	A	C	A			
	1	B		B			
	2	D		C			
				D			
		MOB(C) > MThr		MOB(C) < MThr			

LUCAS Mobility Example



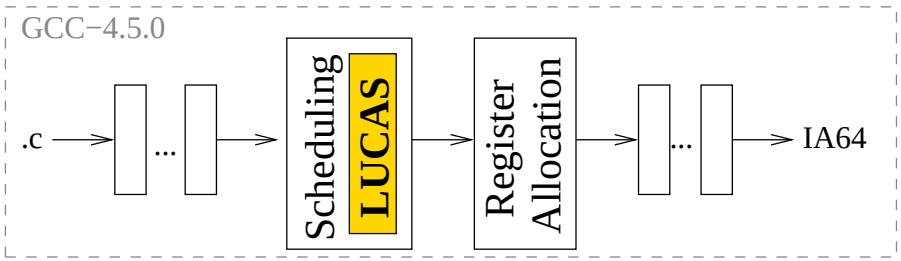
Data Flow Graph

		1 CYCLE		2 CYCLES		3 CYCLES	
		CL0	CL1	CL0	CL1	CL0	CL1
SC	0	A	C	A	C	A	C
	1	B		B		B	
	2	D					
				D			
						D	
CC	0	A		A		A	
	1	B		B		B	
	2	C		C		C	
	3	D		D		D	
LUCAS	0	A	C	A		A	
	1	B		B		C	
	2	D		C		B	
				D		D	
		MOB(C) > MThr		MOB(C) < MThr		MOB(C) < MThr	

Outline

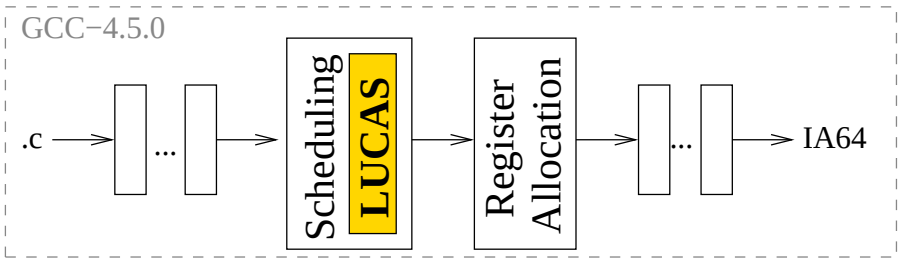
Experimental Setup

- Compiler: GCC-4.5.0, Modified Haifa-Scheduler



Experimental Setup

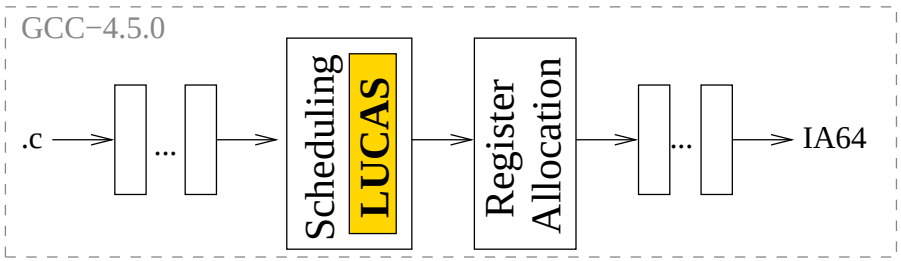
- Compiler: GCC-4.5.0, Modified Haifa-Scheduler



- Architecture
  - IA64-based 4-cluster, 4/8-issue VLIW 1-4cycle delay
  - Modified SKI IA64 simulator

Experimental Setup

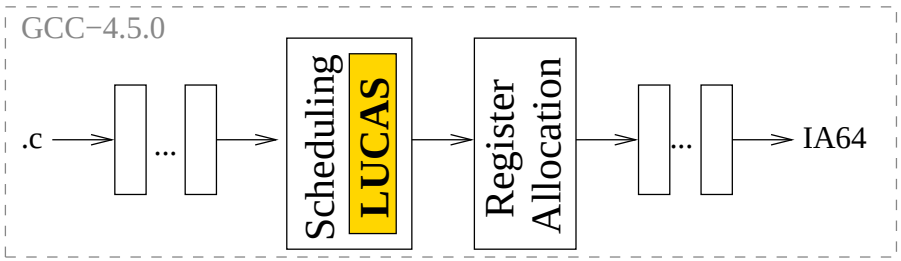
- Compiler: GCC-4.5.0, Modified Haifa-Scheduler



- Architecture
  - IA64-based 4-cluster, 4/8-issue VLIW 1-4cycle delay
  - Modified SKI IA64 simulator
- Benchmarks: MediabenchII Video Benchmark suite

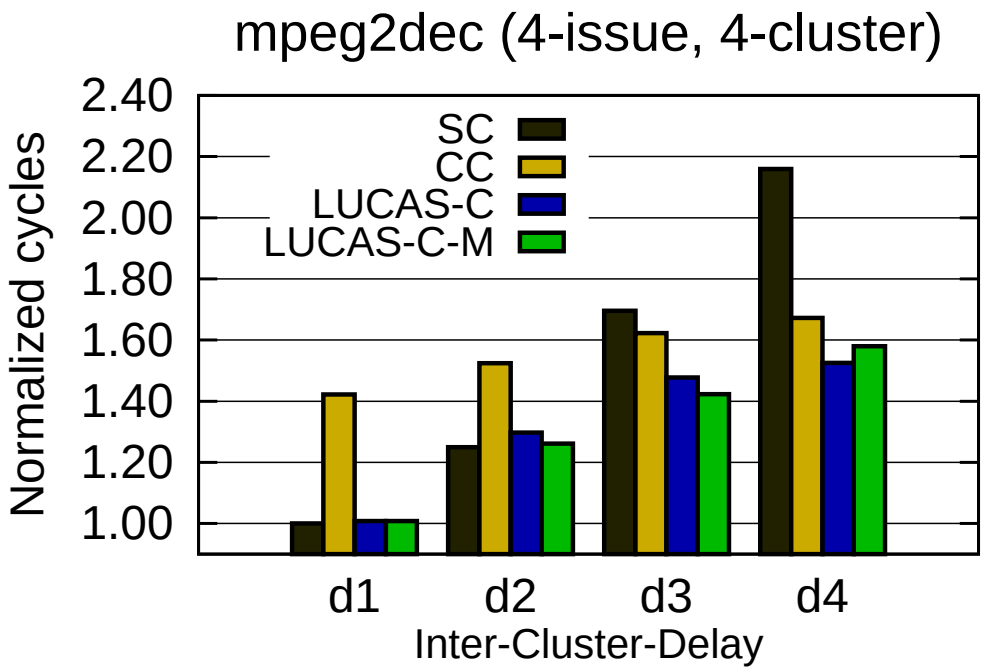
Experimental Setup

- Compiler: GCC-4.5.0, Modified Haifa-Scheduler



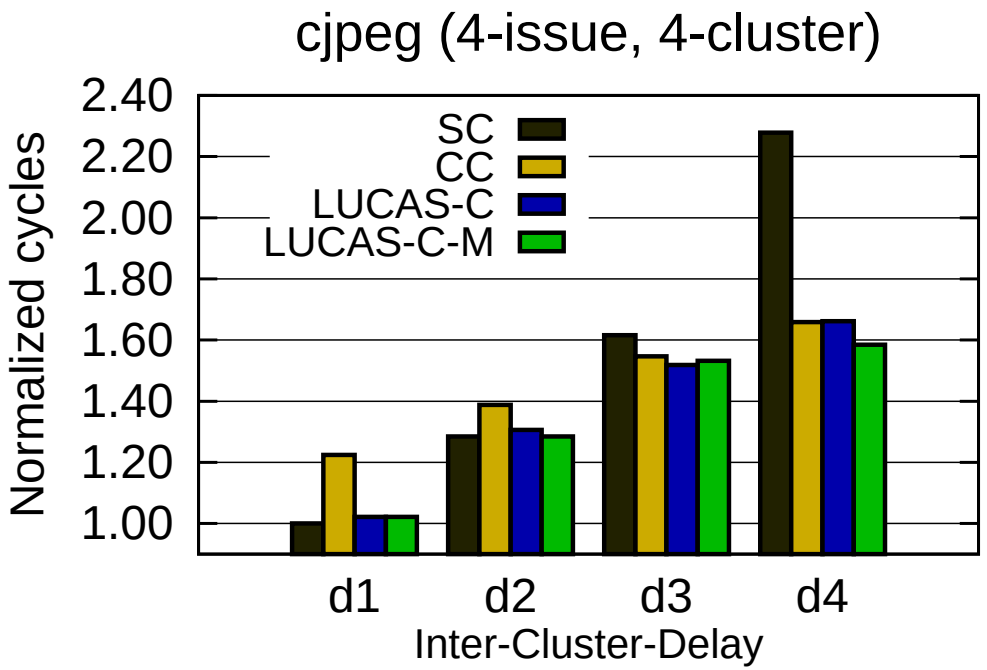
- Architecture
  - IA64-based 4-cluster, 4/8-issue VLIW 1-4cycle delay
  - Modified SKI IA64 simulator
- Benchmarks: MediabenchII Video Benchmark suite
- Compare:
  - Start-Cycle (SC), Completion-Cycle (CC), LUCAS with Congestion (LUCAS-C) and Mobility (LUCAS-C-M)
  - For comparisons against UAS and Critical-Successor (CS), please refer to the paper.

Performance Comparison



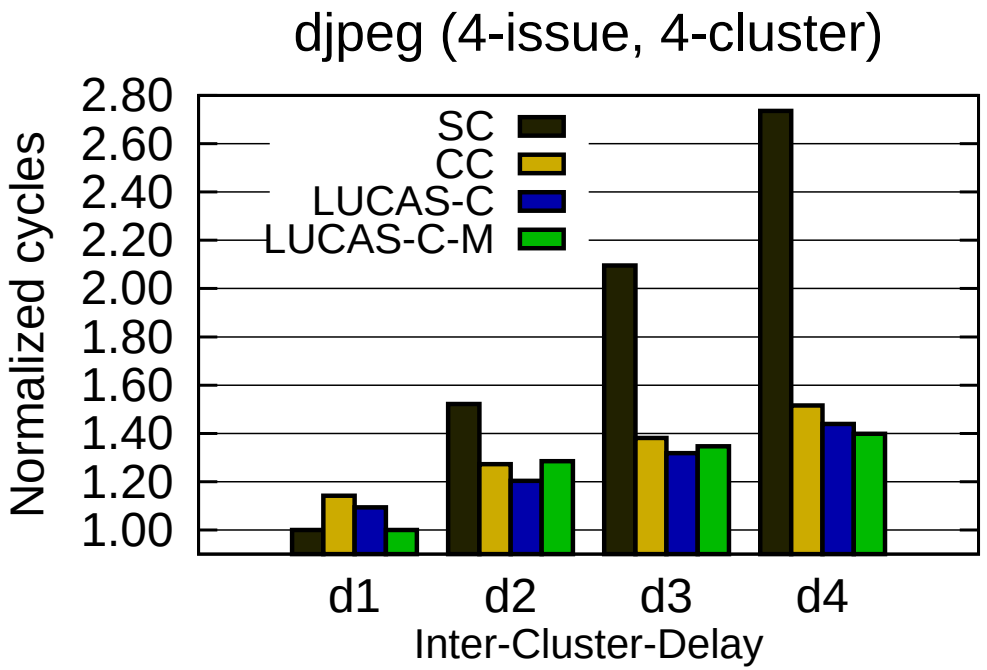
- Match SC performance at low inter-cluster delays
- Match CC at high delays
- Often outperform best due to fine-grain switching

Performance Comparison



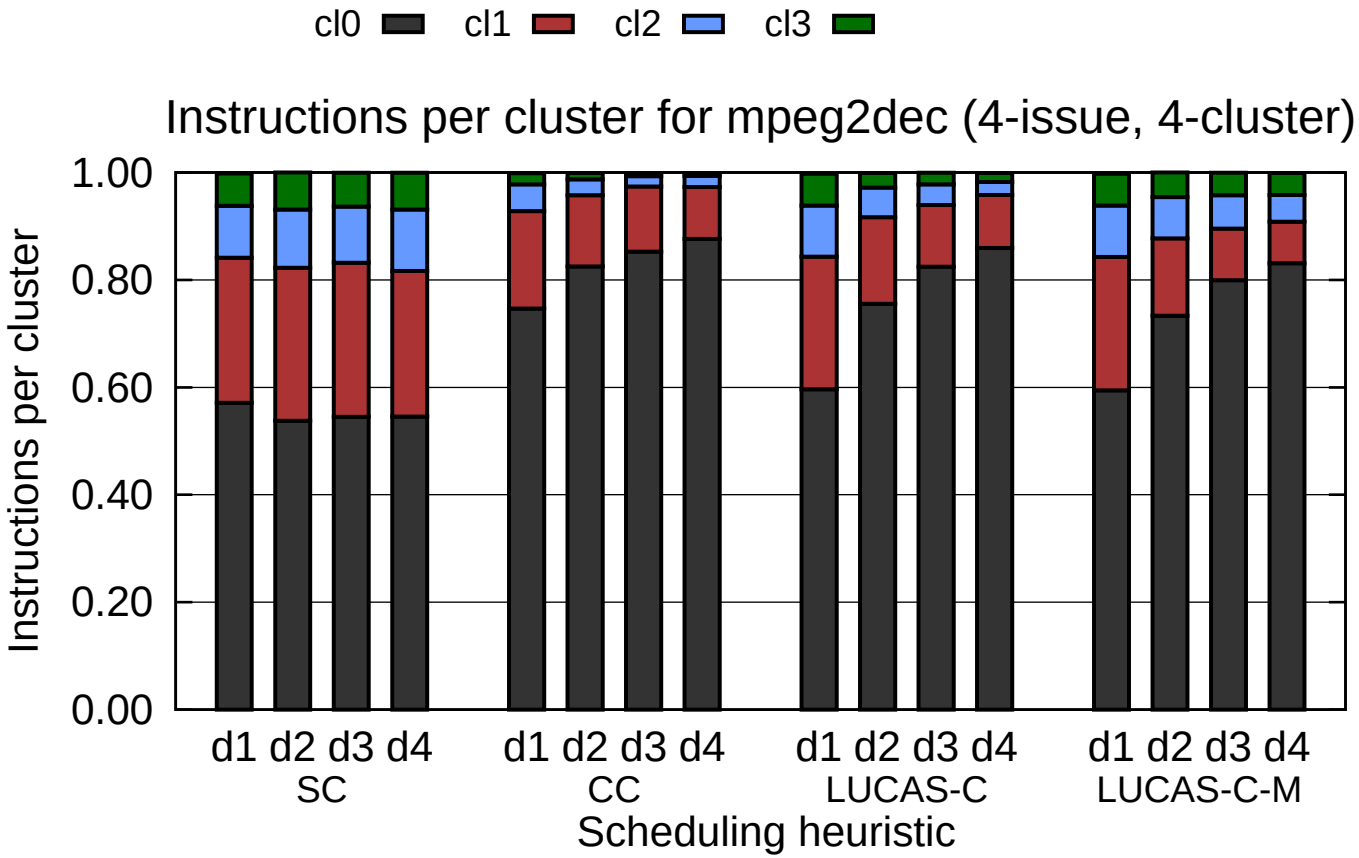
- Match SC performance at low inter-cluster delays
- Match CC at high delays
- Often outperform best due to fine-grain switching
- Similar results in most benchmarks

Performance Comparison



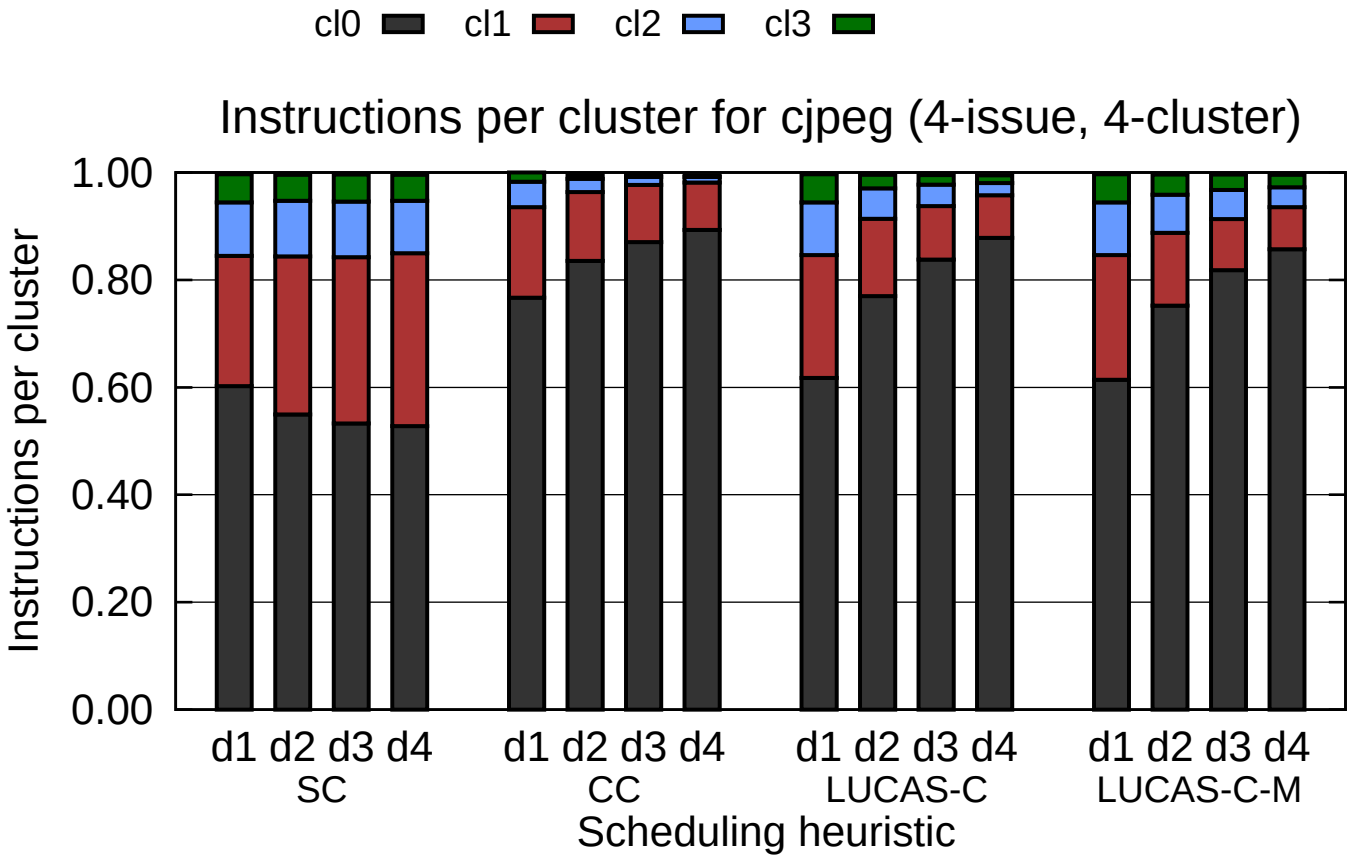
- Match SC performance at low inter-cluster delays
- Match CC at high delays
- Often outperform best due to fine-grain switching
- Similar results in most benchmarks

Instruction Distribution



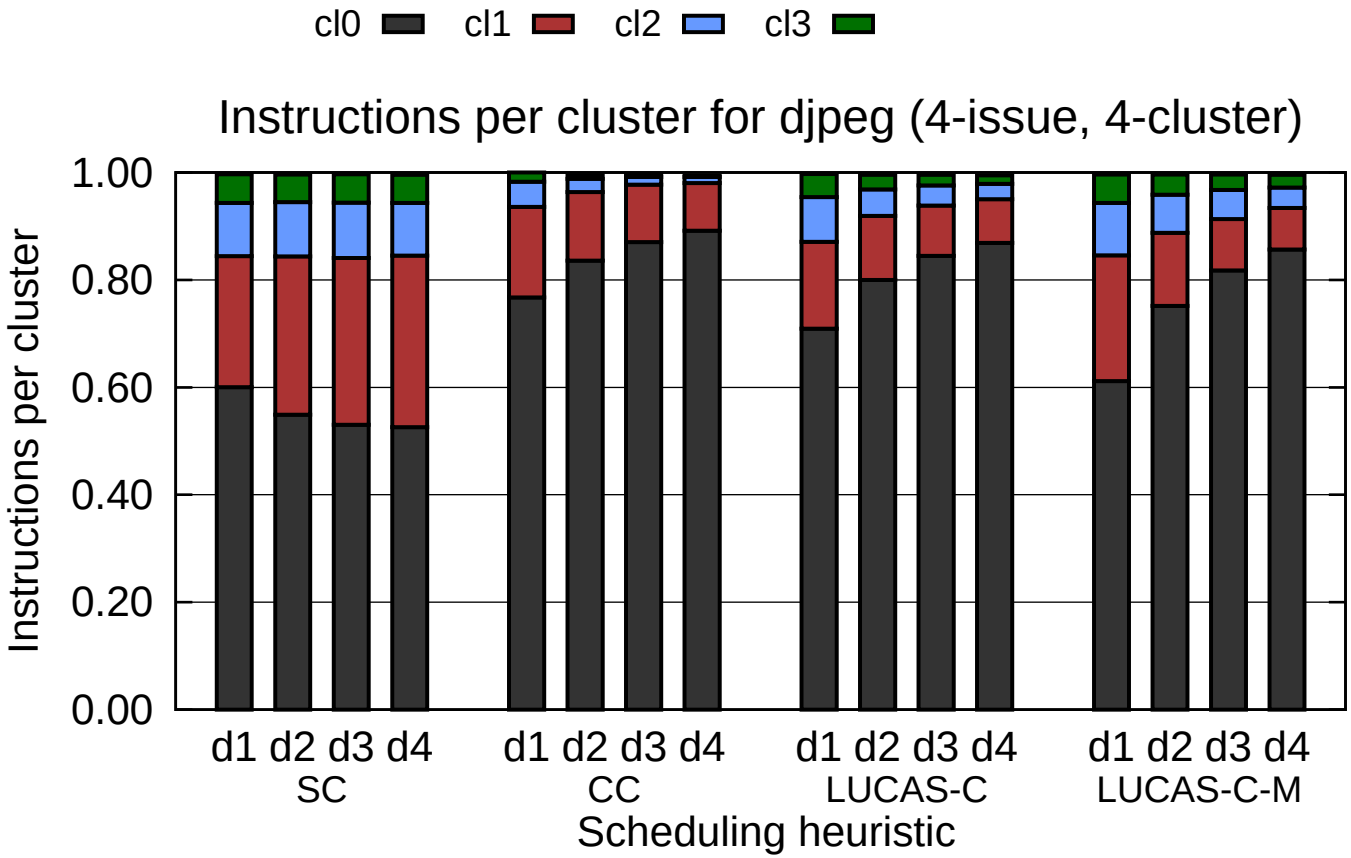
- LUCAS: Aggressive distribution at small delays.
- Less aggressive at high delays

Instruction Distribution



- LUCAS: Aggressive distribution at small delays.
- Less aggressive at high delays
- Similar results in most benchmarks

Instruction Distribution



- LUCAS: Aggressive distribution at small delays.
- Less aggressive at high delays
- Similar results in most benchmarks

Conclusion

Proposed LUCAS, a Latency-Adaptive scheduler for clustered VLIWs that:

- Performs unified cluster assignment and instruction scheduling
- Generates fast code no matter the inter-cluster delay
- Often outperforms the best of the state-of-the-art due to fine-grain switching between heuristics.

LUCAS: Latency-adaptive Unified Cluster Assignment and instruction Scheduling

Vasileios Porpodas and Marcelo Cintra

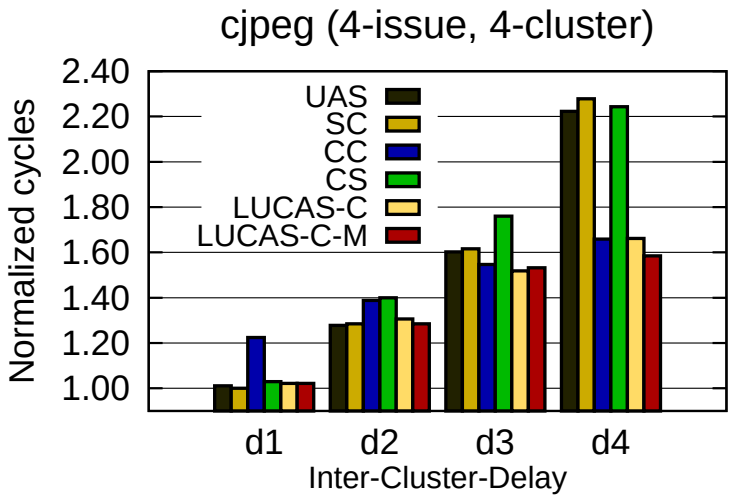
University of Edinburgh

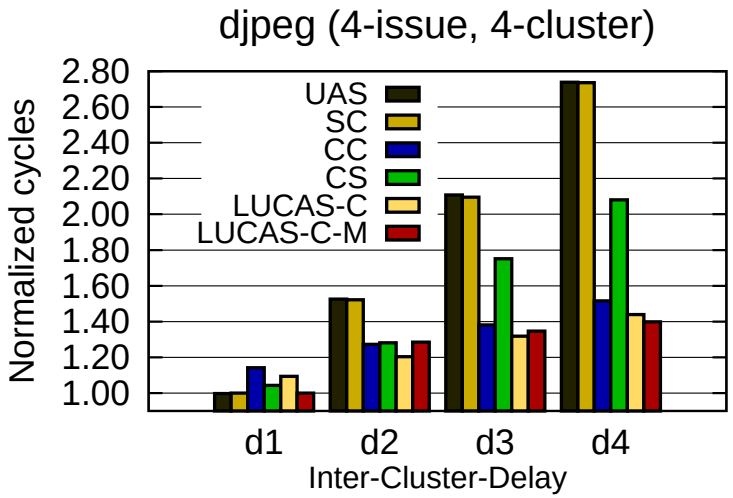
LCTES 2013

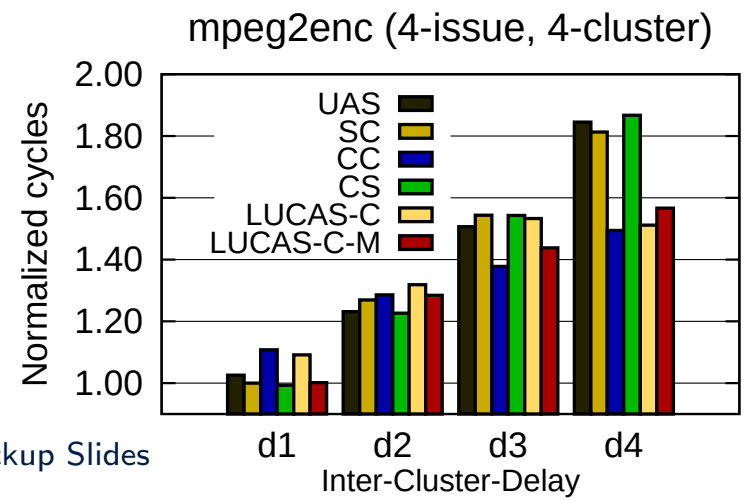
Backup slides

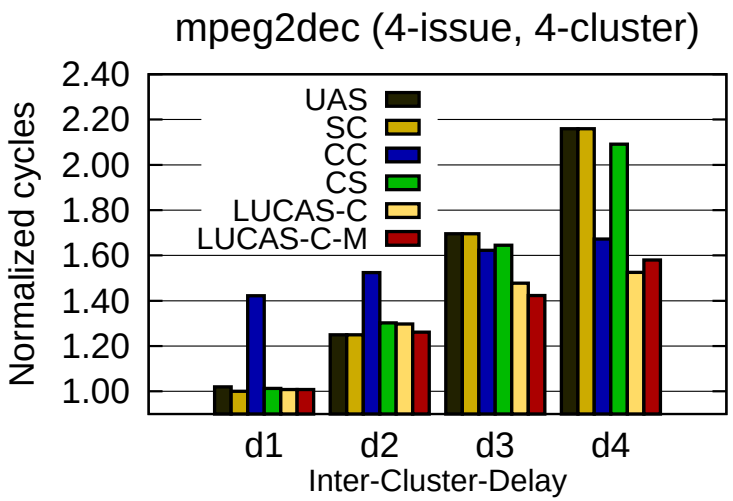
- Performance results, issue4
- Performance results, issue8
- Instruction distribution, issue4
- Instruction distribution, issue8
- Processor Configuration
- Congestion Threshold
- Mobility Threshold
- Definition of Mobility
- LUCAS Algorithm 1
- LUCAS Algorithm 2
- Bibliography

Performance Results (issue 4)



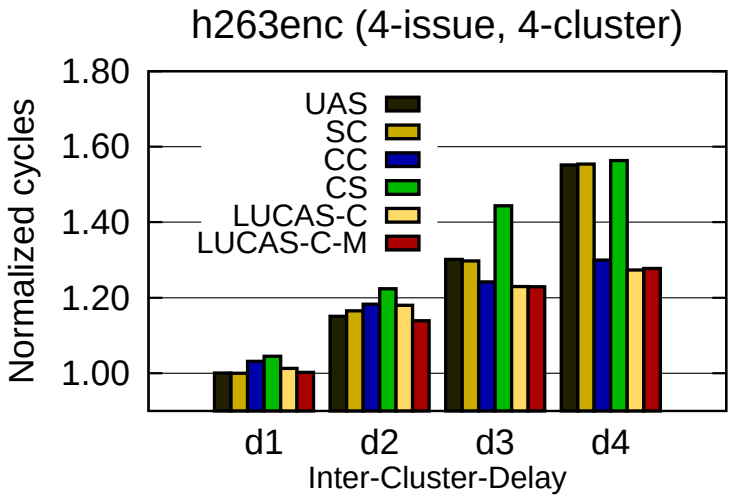


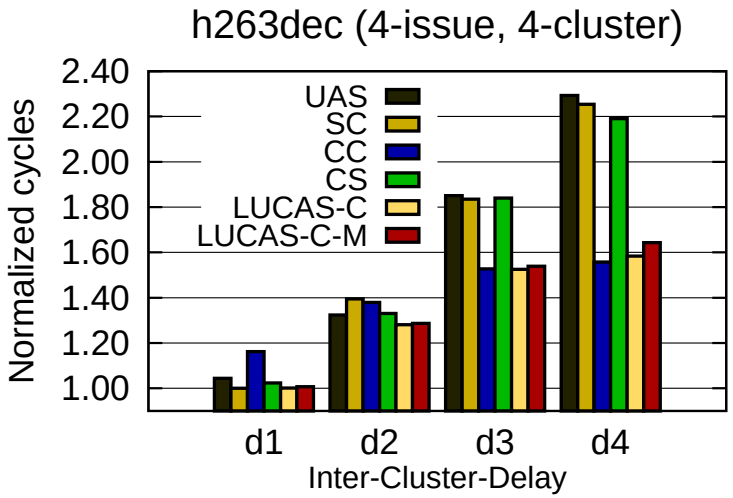




Backup Slides

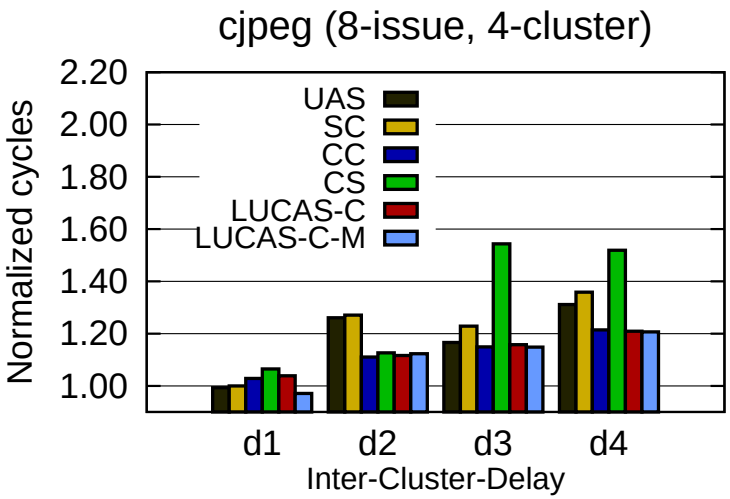
Performance Results (issue 4)

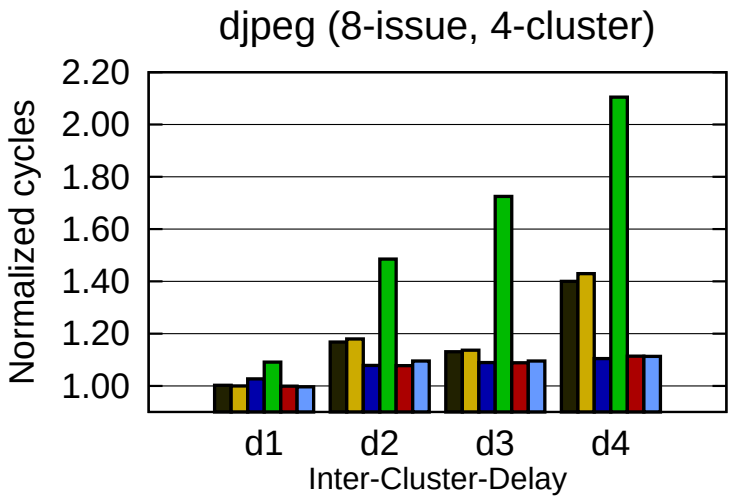


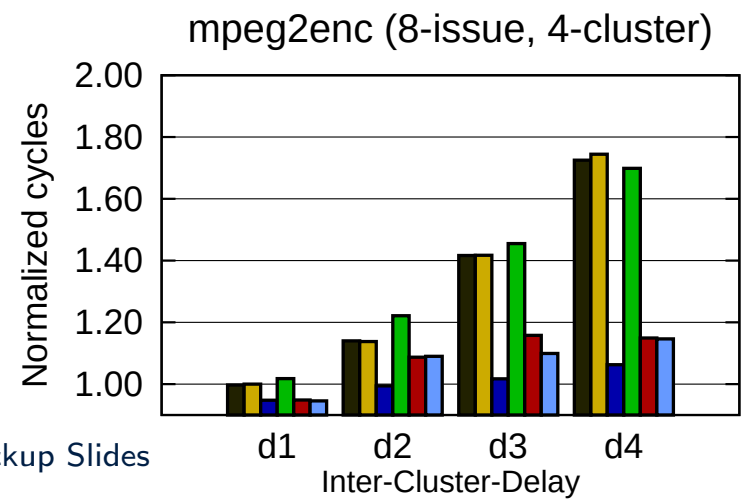


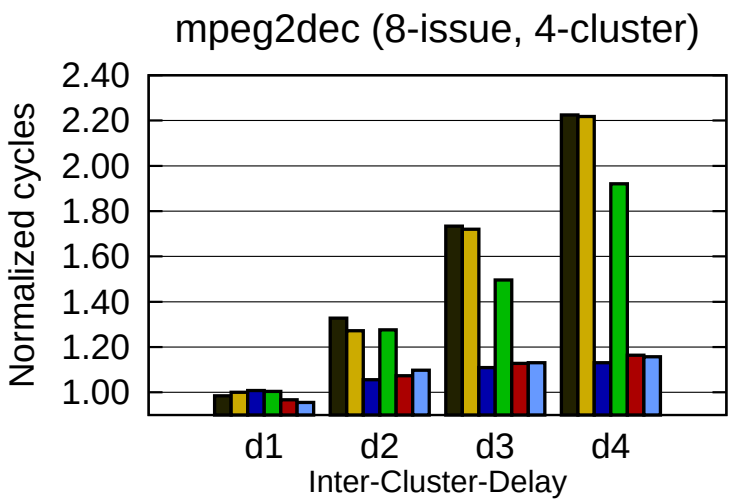
Backup Slides

Performance Results (issue 8)



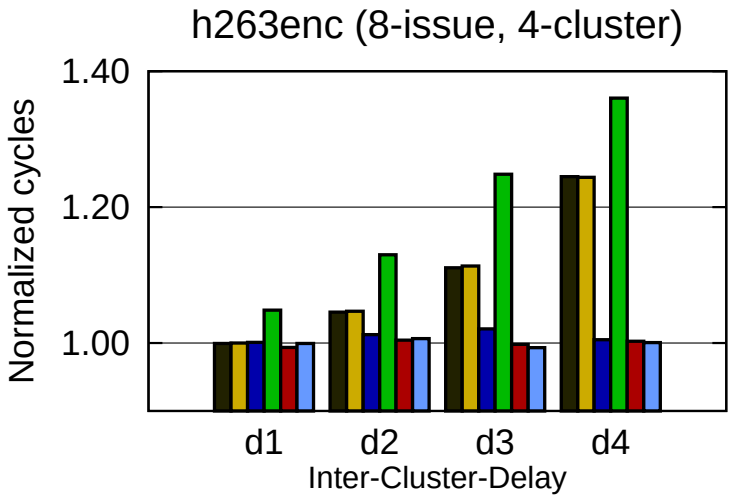


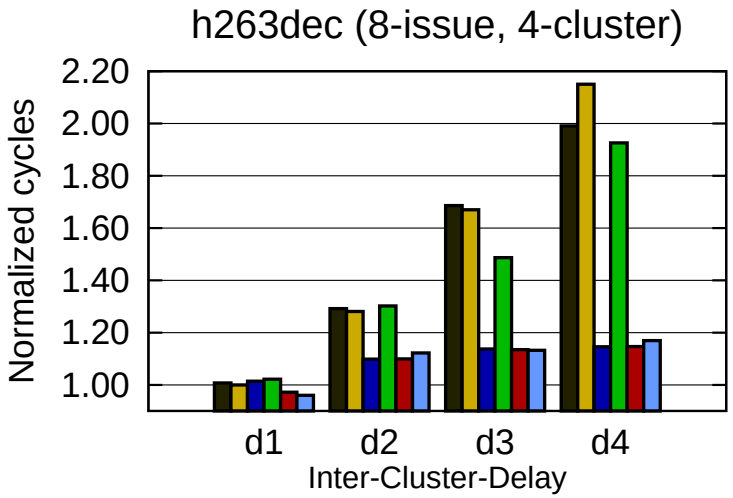




Backup Slides

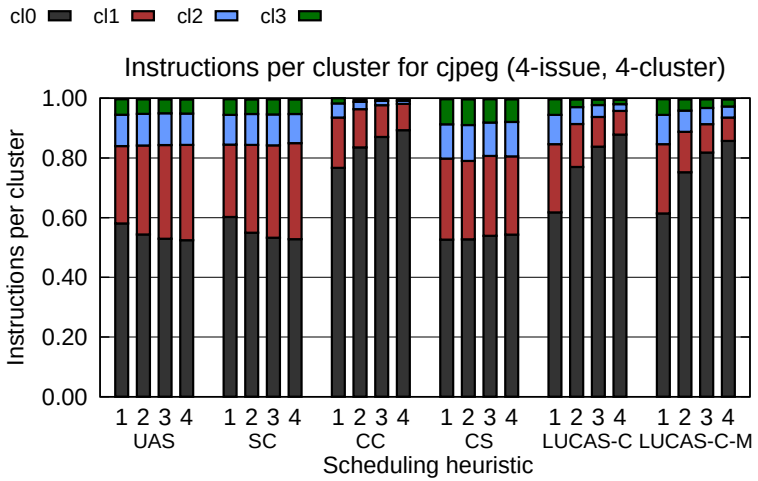
Performance Results (issue 8)

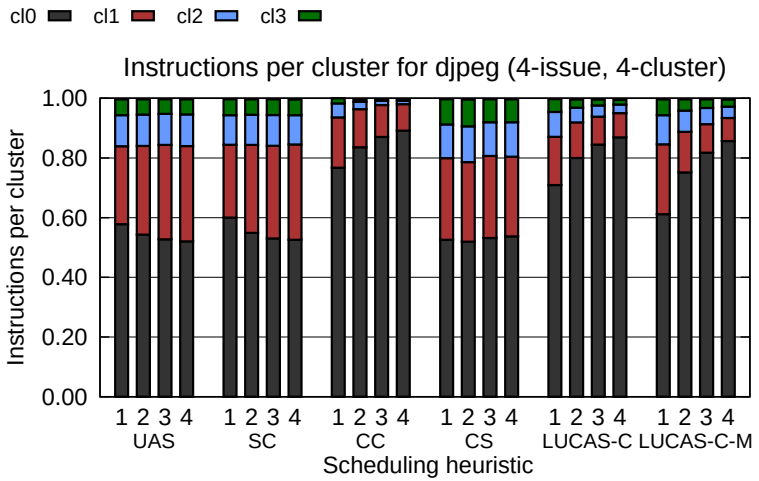


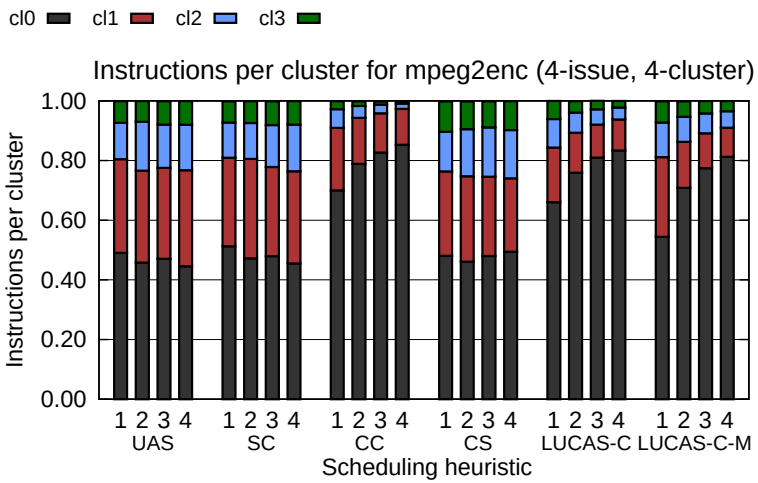


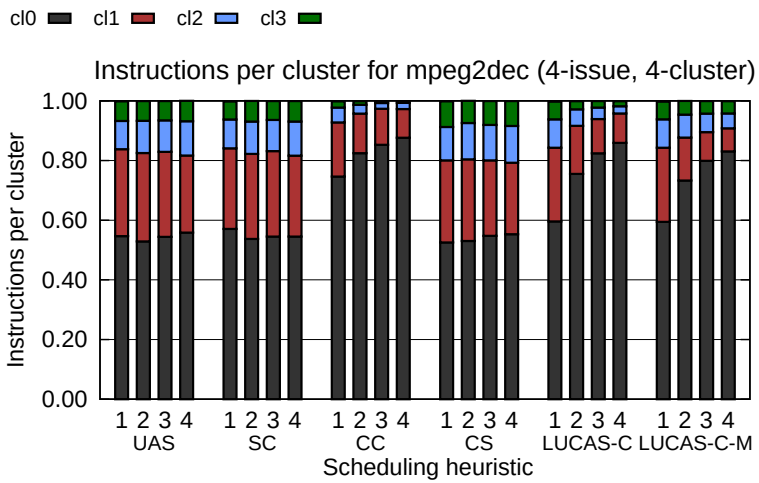
Backup Slides

Instruction Distribution (issue 4)



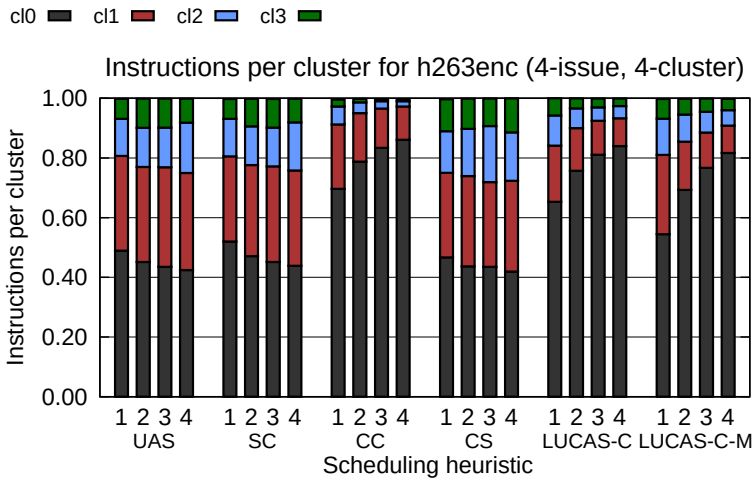


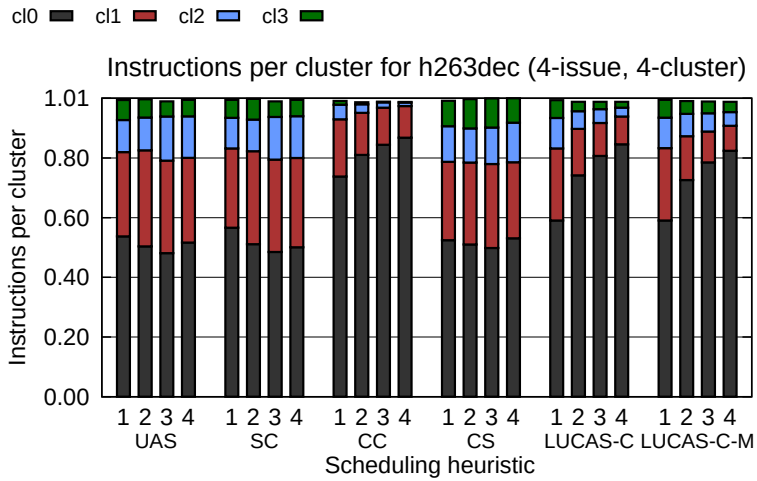




Backup Slides

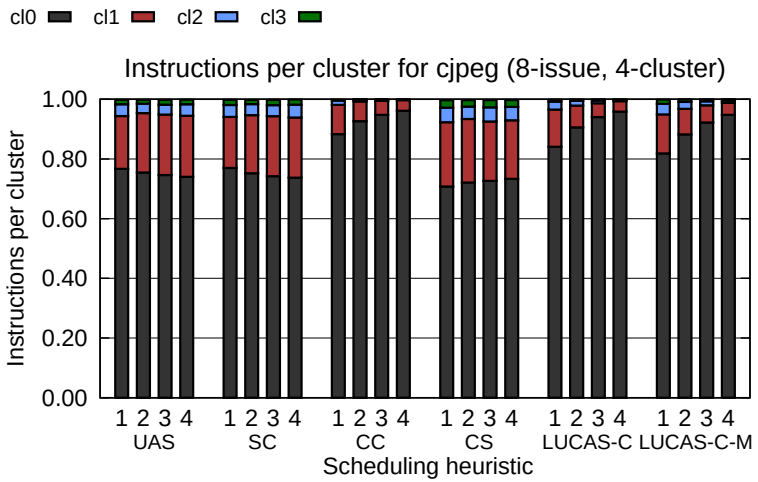
Instruction Distribution (issue 4)

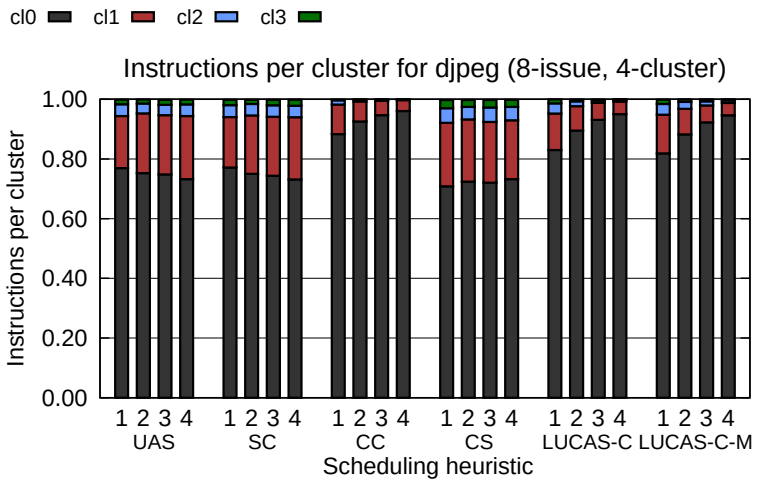


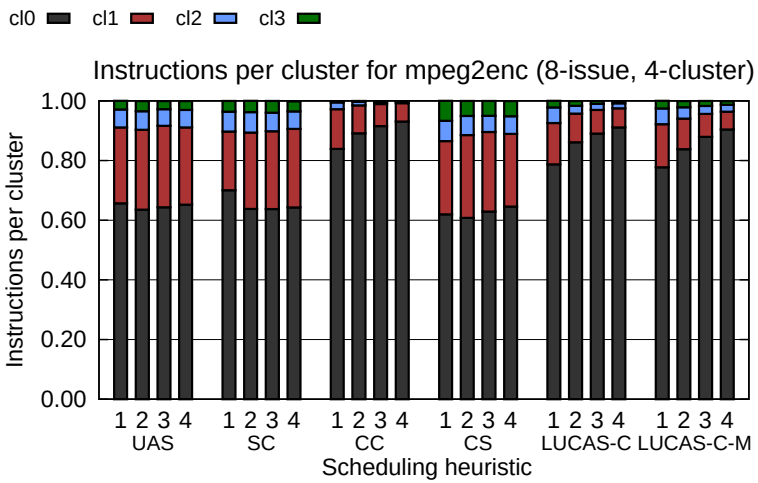


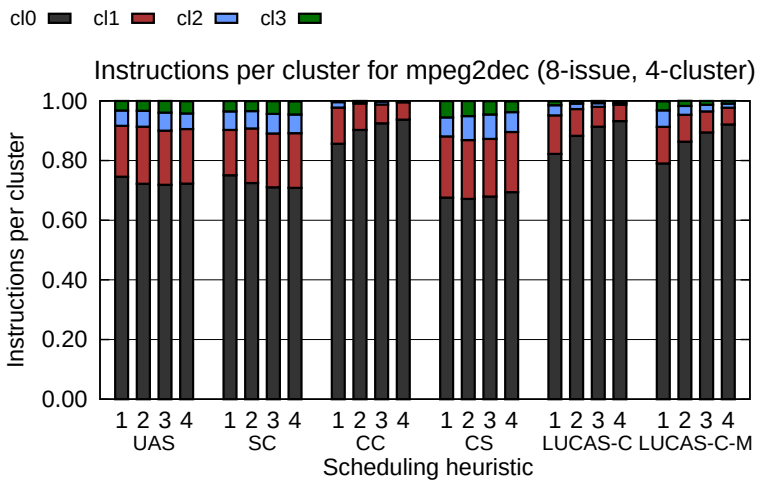
Backup Slides

Instruction Distribution (issue 8)



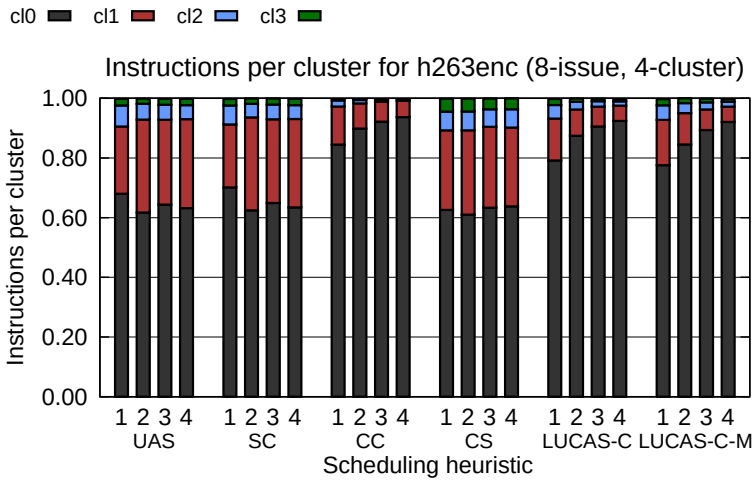


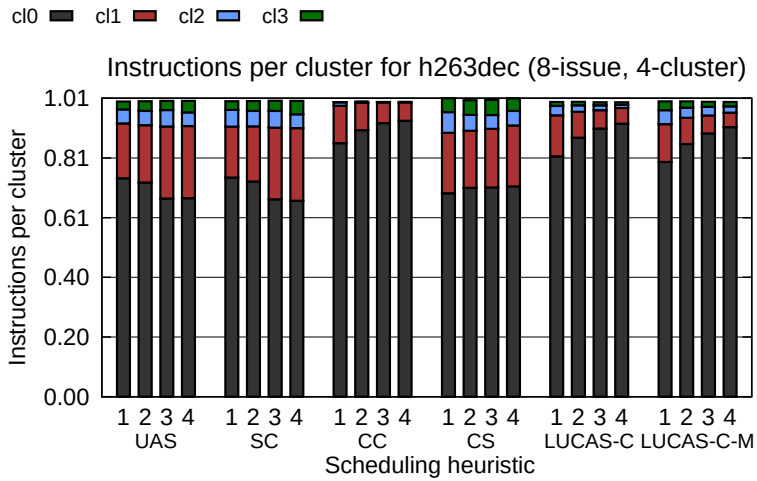




Backup Slides

Instruction Distribution (issue 8)





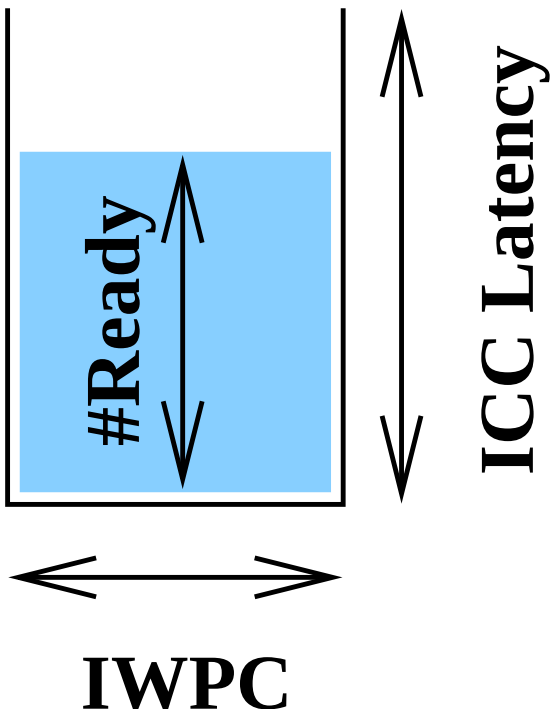
Backup Slides

Processor: IA64 based clustered VLIW				
Issue Width:	4 or 8			
Clusters:	4			
Instruction Latencies:	Same as Itanium2			
Register File:	(32GP, 32FL, 16PR) per cluster			
Inter-Cluster Delay:	1 - 4 cycles			
Inter-Cluster Bus Bandwidth:	∞			
Branch Prediction:	Perfect			
Cache: Levels 3 (same as Itanium2)				
Levels :	L1	L2	L3	Main Mem.
Size (Bytes):	16K	256K	3M	∞
Block size (Bytes):	64	128	128	-
Associativity:	4-Way	8-way	12-way	-
Latency (cycles):	1	5	12	150

Table: Processor configuration.

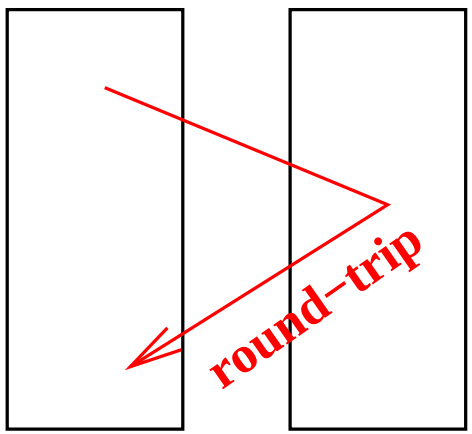
High Congestion Threshold

- High Congestion =
  Num. Ready instructions (cycle) $>$ CThr
- CThr = IWPC $\times$ ICC Latency



High Mobility Threshold

- High Mobility = (Mobility (insn) > MThr)
  MThr = IWPC × 2 × (ICC-Latency - 1)
- 2 × (ICC-Latency - 1) : is the round-trip latency



2 X (ICC−Latency − 1)

- IWPC × : Increases penalty as issue-width increases

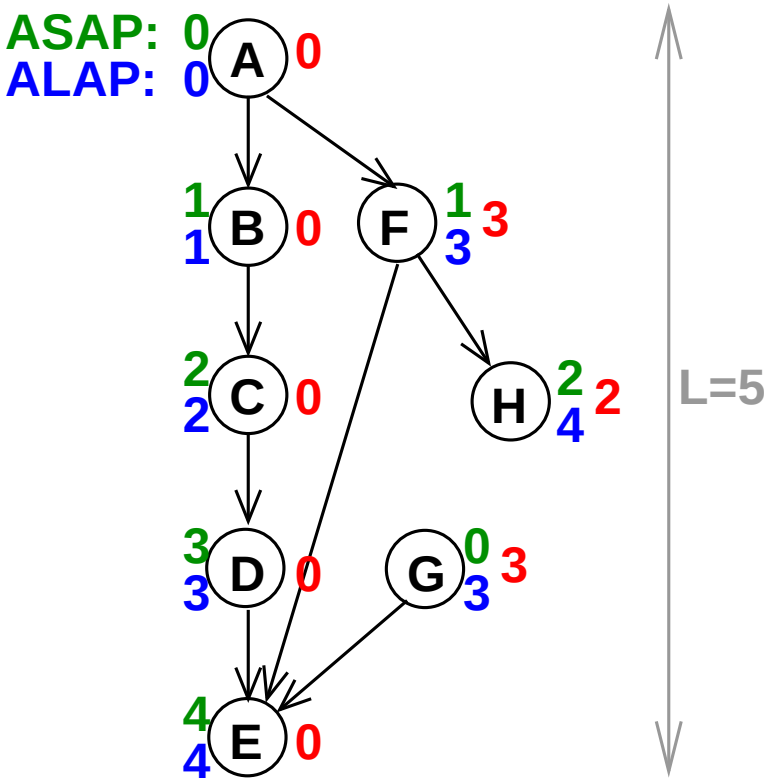
Algorithm

```
 1 /* LUCAS Scheduling and clustering. */
 2 lucas_schedule_and_cluster (DFG)
 3 {
 4  walk down the DFG and prioritize the nodes
 5  cycle = 0
 6  while (exist unscheduled nodes)
 7    Fill in ready_list
 8    sort ready_list based on priority
 9    for node in prioritized ready_list
10      best_cl = get_best_cluster(node.instr, cycle)
11      if (start_cycle (node.instr, cluster)<=cycle)
12        if (can issue node.instr on cycle)
13          if (can schedule Inter-Cluster Copies)
14            Emit ICCs and reg. rename node.instr
15            node.cluster = best_cl
16            Issue (node.instr, cycle, best_cl)
17            Update MOBILITY(node.instr) if it gets data from a
                  ↪distant cluster
18    cycle ++
19 }
```

```
 1 get_best_cluster (insn, cycle)
 2 {
 3  for cluster in all clusters
 4    heuristic[cluster] = lucas(insn,cluster)
 5  /* Find best cluster: MIN_CL */
 6  min_cl = 0
 7  for cli in clusters
 8    if (heuristic[cli] < heuristic[min_cl])
 9      min_cl = cli
10  return min_cl
11 }
12
13 /* Return the score of CLUSTER */
14 lucas (insn, cluster)
15 {
16  high_congestion = (#Ready-instr. > IWPC × ICD)
17  high_mobility = (MOBILITY(insn) > IWPC×2×(ICD-1))
18  if (high_congestion OR high_mobility)
19    return start_cycle (insn, cluster)
20  else
21    return completion_cycle (insn, cluster)
22 }
```

Instruction Mobility

MOBILITY = ALAP − ASAP



ASAP "as soon as possible"
the earliest scheduling cycle

ALAP "as late as possible"
the latest possible scheduling cycle
such that a valid schedule that
completes in L cycles is feasible

on a datapath with infinite resources.

Backup Slides

LUCAS vs UAS Complexity

Algorithm	Worst-case	Practical
UAS	$O(N^3)$	$O(N)$
LUCAS	$O(N^3)$	$O(N)$

LUCAS and UAS have a similar 3-nested loop structure and exhibit similar complexity. For all practical cases, the complexity is $O(N)$ and the worst-case is $O(N^3)$.

Bibliography

SC, CC Heuristics:

J. Ellis.
Bulldog: A compiler for vliw architectures.
Technical report, Yale Univ., 1985.

UAS Algorithm:

E. Ozer, S. Banerjia, and T. Conte.
Unified assign and schedule: a new approach to scheduling for clustered register file microarchitectures.
In *MICRO*, 1998.

CS Heuristic:

X. Zhang, H. Wu, and J. Xue.
An efficient heuristic for instruction scheduling on clustered vliw processors.
In *CASES*, 2011.

Mobility:

V.S. Lapinskii, M.F. Jacome, and G.A. De Veciana.
Cluster assignment for high-performance embedded VLIW processors.
In *TODAES*, 2002.

Backup Slides