Constraint Programming in Compiler Optimization: Lessons Learned

Peter van Beek
University of Waterloo

Acknowledgements

- Joint work with:
  - Omer Beg
  - Alejandro López-Ortiz
  - Abid Malik
  - Jim McInnes
  - Wayne Oldford
  - Claude-Guy Quimper
  - John Tromp
  - Kent Wilken
  - Huayue Wu
- Funding:
  - NSERC
  - IBM Canada

Application-driven research

- Idea:
    - pick an application—a real-world problem—where, if you solve it, there would be a significant impact
- Along the way, if all goes well, you will also:
    - identify and fill gaps in theory
    - identify and solve interesting sub-problems whose solutions will have general applicability

Optimization problems in compilers

- Instruction selection
- Instruction scheduling
    - basic-block scheduling
    - super-block scheduling
    - loop scheduling: tiling, unrolling, fusion
- Memory hierarchy optimizations
- Register allocation

Optimization problems in compilers

- Instruction selection
- Instruction scheduling
    - basic-block scheduling
    - super-block scheduling
    - loop scheduling: tiling, unrolling, fusion
- Memory hierarchy optimizations
- Register allocation

Production compilers

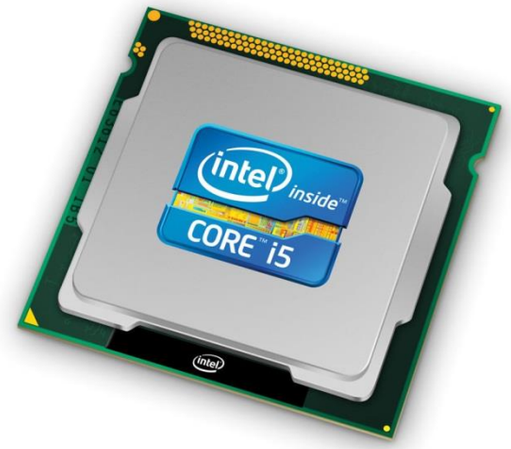
> “At the outset, note that basic-block scheduling is an NP-hard problem, even with a very simple formulation of the problem, so we must seek an effective heuristic, rather than an exact approach.”
>
> Steven Muchnick, *Advanced Compiler Design & Implementation*, 1997

Outline

- Introduction
    - computer architecture
    - superblock scheduling
- Constraint programming approach
    - temporal scheduler
    - spatial and temporal scheduler
- Experiments
    - experimental setup
    - experimental results
- Lessons learned

Computer architecture: Performing instructions in parallel

- Multiple-issue
    - multiple functional units; e.g., ALUs, FPUs, load/store units, branch units
    - multiple instructions can be issued (begin execution) each clock cycle
    - *issue width:* max number of instructions that can be issued each clock cycle
    - on most architectures issue width less than number of functional units



Computer architecture: Performing instructions in parallel

- Pipelining
    - overlap execution of instructions on a single functional unit
    - *latency* of an instruction

      number of cycles before result is available
- *execution time* of an instruction

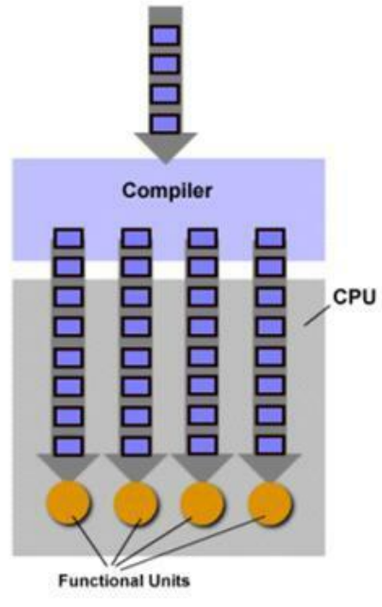
  number of cycles before next instruction can be issued on same functional unit
- *serializing instruction*

  instruction that requires exclusive use of entire processor in cycle in which it is issued

Analogy: vehicle assembly line

Superblock instruction scheduling

- Instruction scheduling
    - assignment of a clock cycle to each instruction
    - needed to take advantage of complex features of architecture
    - sometimes necessary for correctness (VLIW)

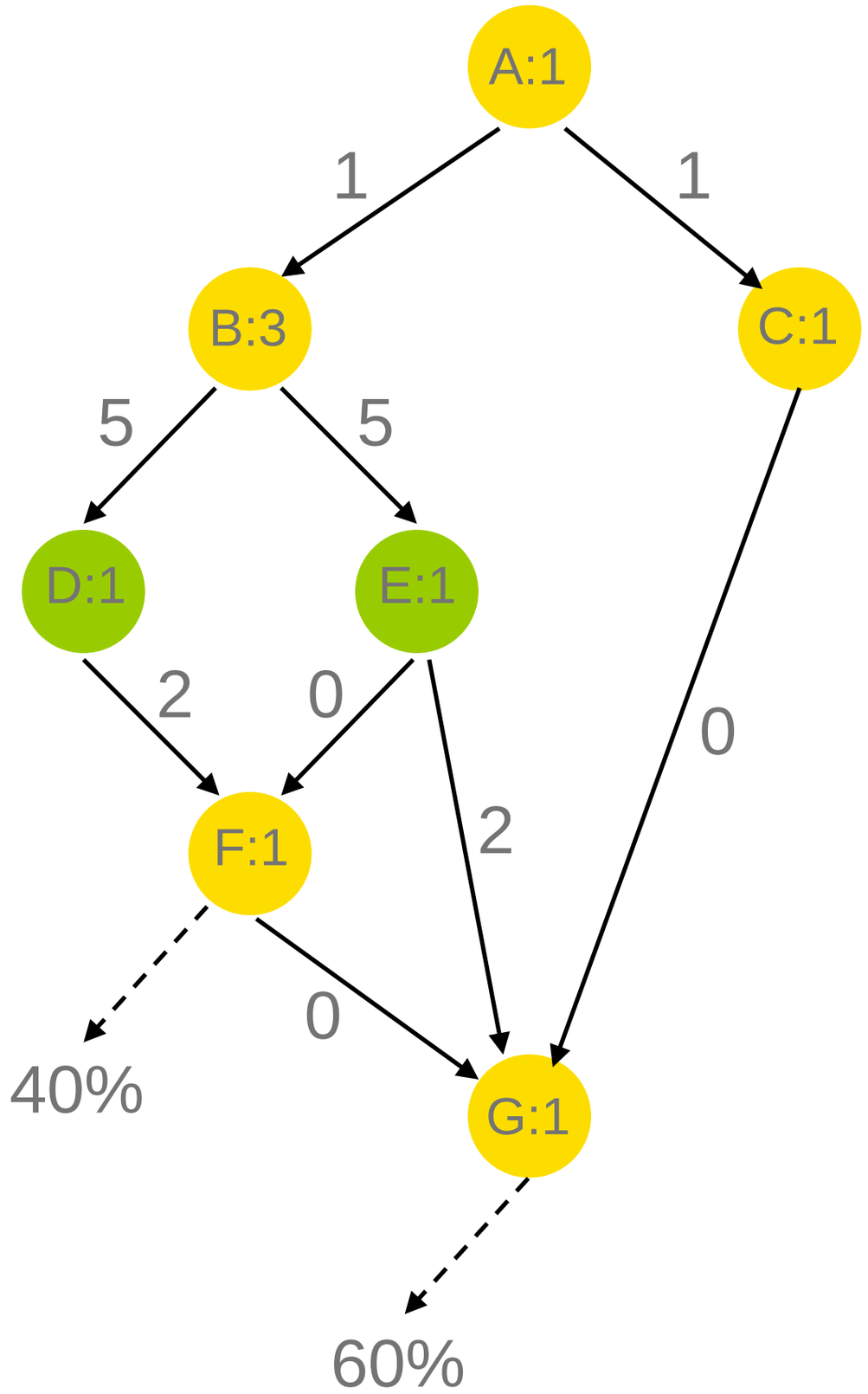


- Basic block
    - straight-line sequence of code with single entry, single exit
- Superblock
    - collection of basic blocks with a unique entrance but multiple exits
- Given a target architecture, find schedule with minimum expected completion time

Example superblock

dependency DAG

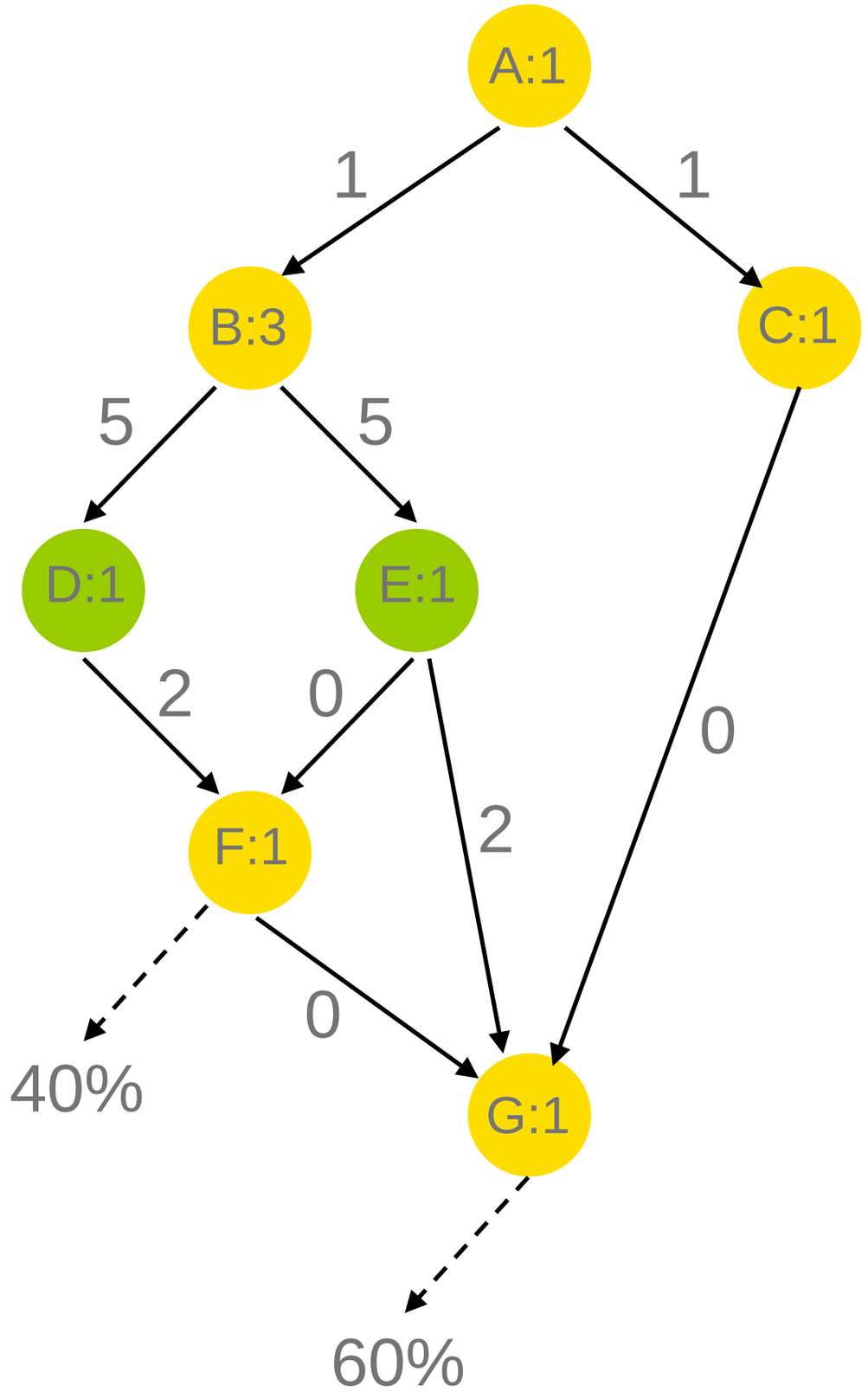
- nodes
  - one for each instruction
  - labeled with execution time
  - nodes F and G are branch instructions, labeled with probability the exit is taken
- arcs
  - represent precedence
  - labeled with latencies



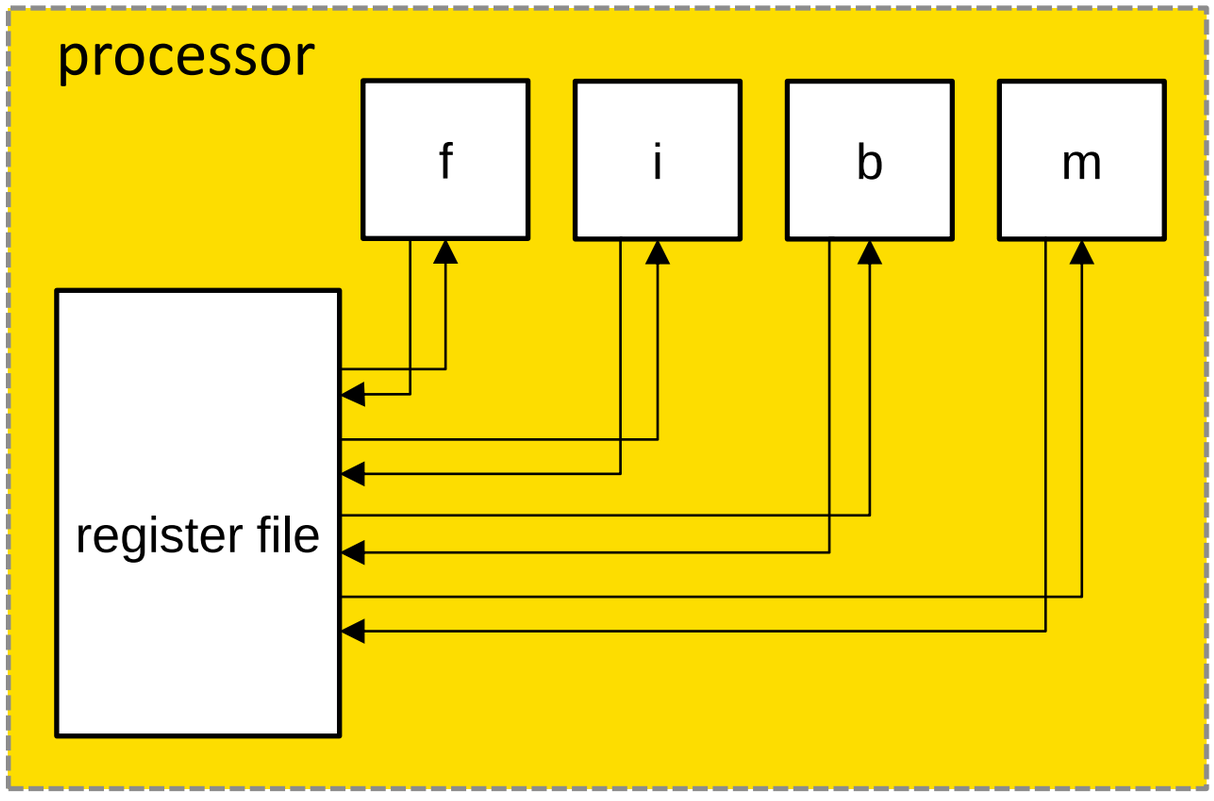
Example superblock

optimal cost schedule for 2-issue processor

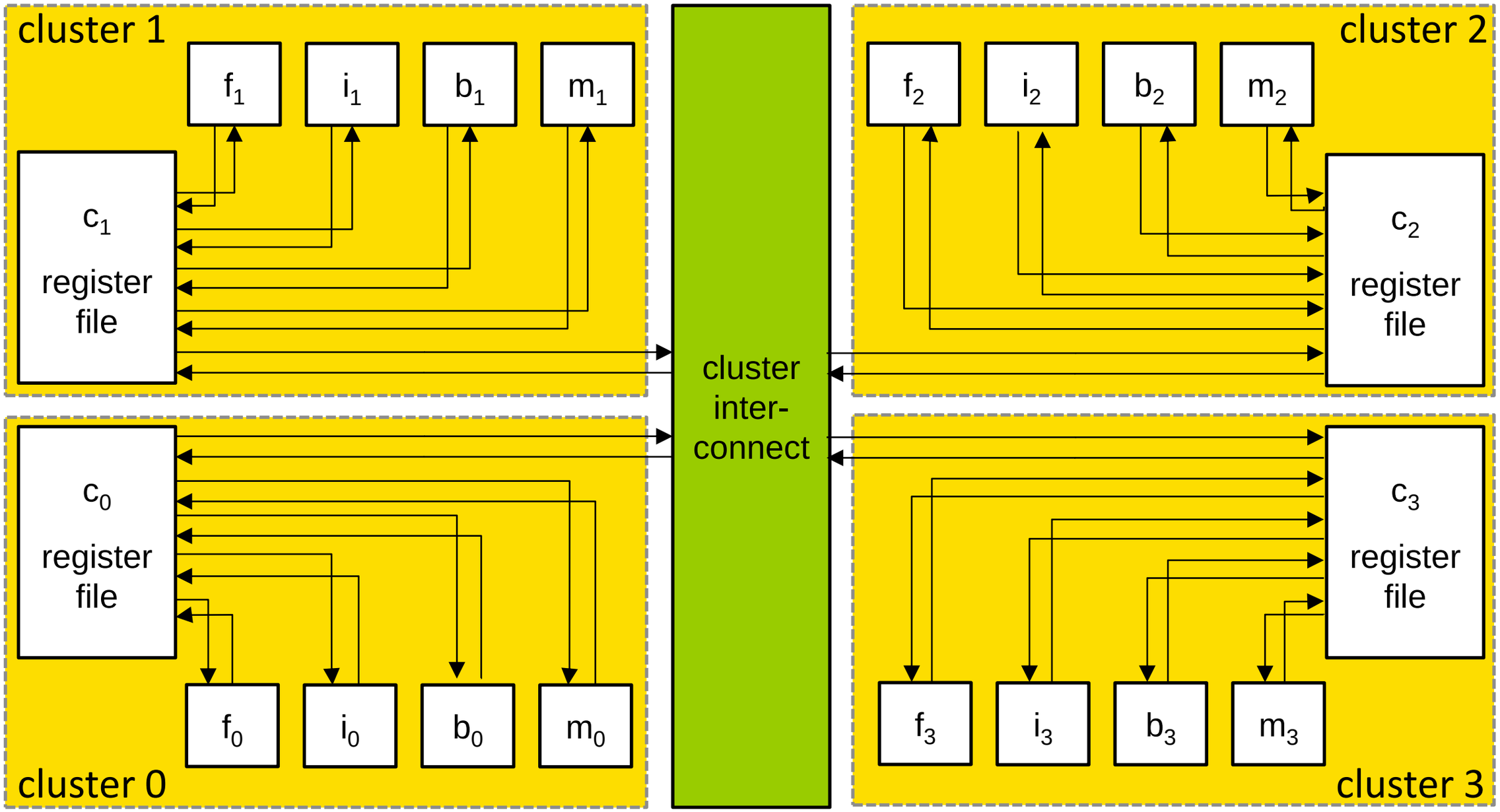
cycle	ALU	FPU
1	A	
2	B	
3		
4		
5	C	
6		
7		D
8		E
9	F	
10	G	



Computer architecture:
General purpose architectures



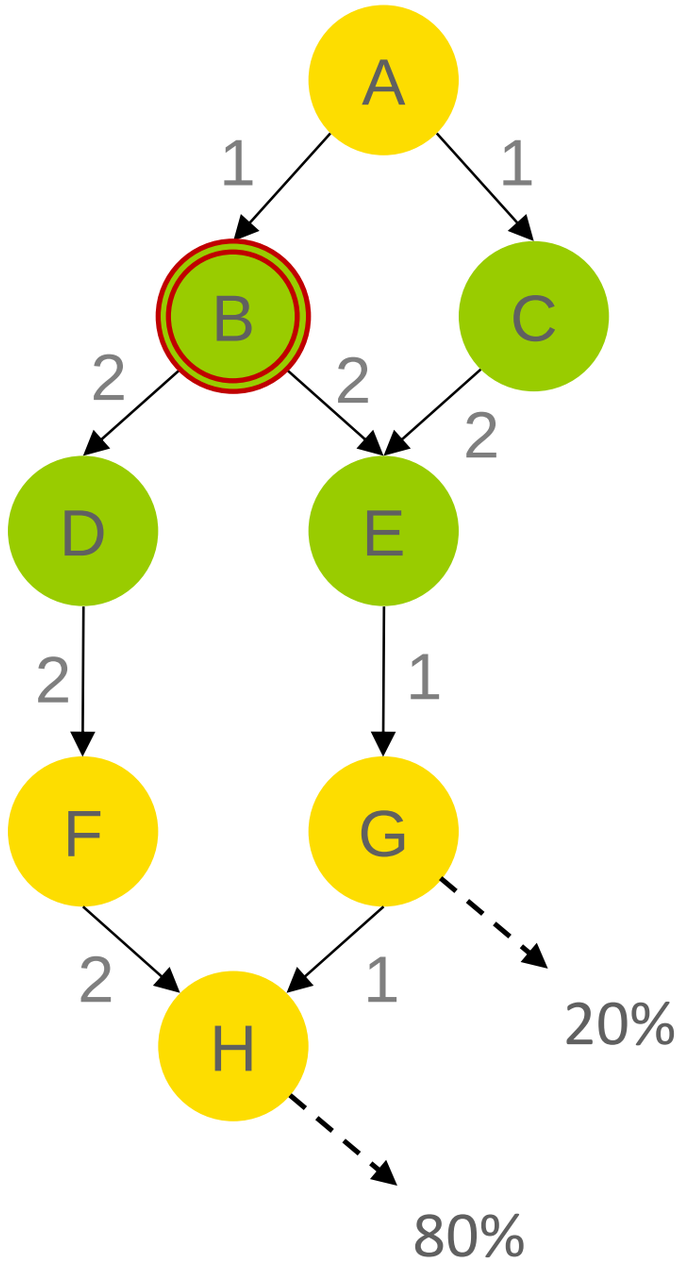
Computer architecture: Clustered architectures



Computer architecture: Clustered architectures

- Current: digital signal processing
    - multimedia, audio processing, image processing
    - wireless, ADSL modems, …
- Future trend: general purpose multi-core processors
    - large numbers of cores
    - fast inter-processor communication

Spatial and temporal scheduling



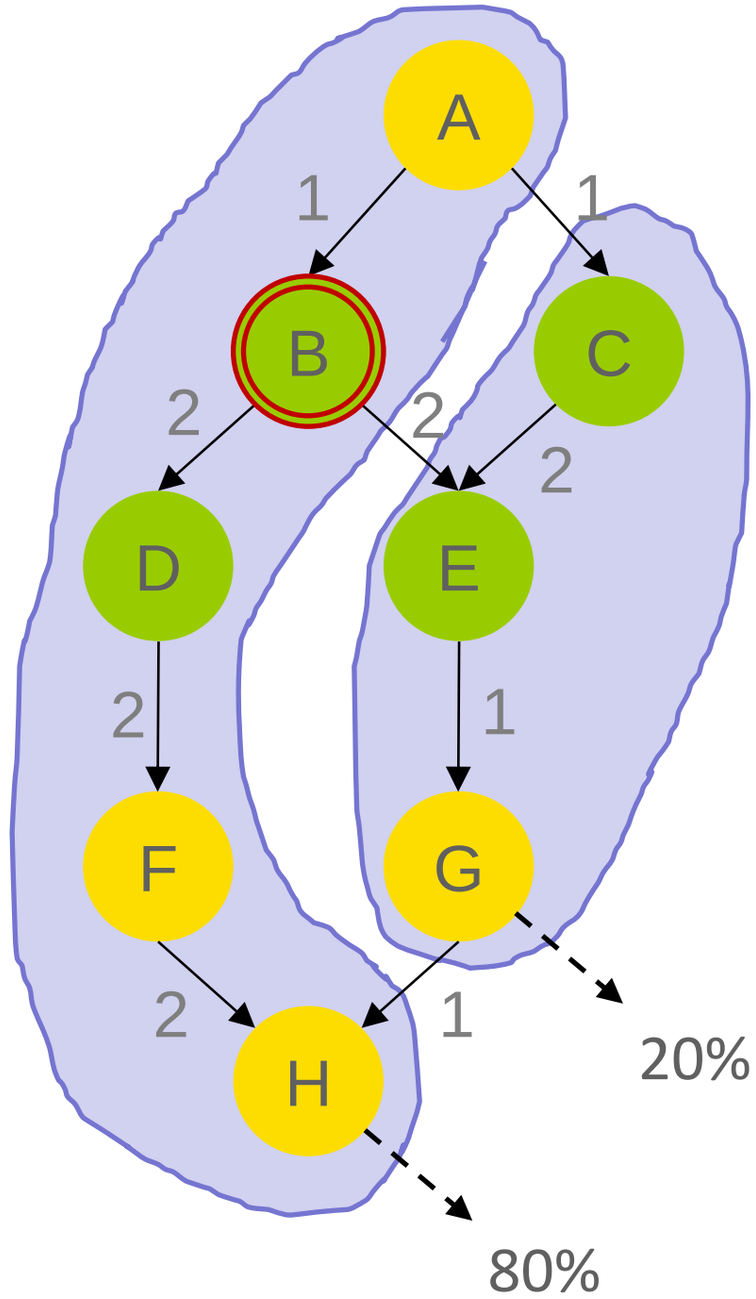
cycle	c0
1	A
2	B
3	
4	C
5	
6	D
7	E
8	F
9	G
10	H

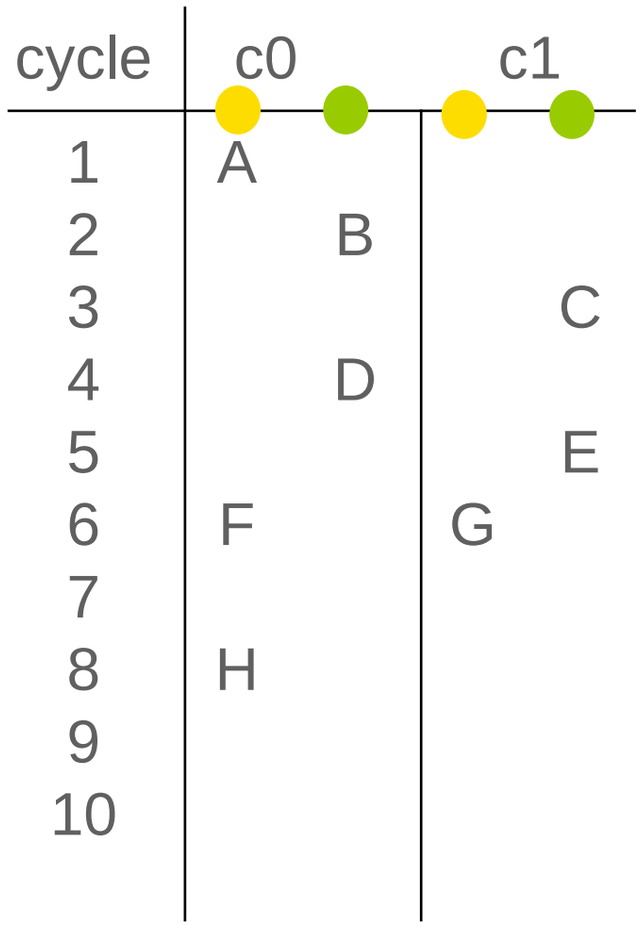
cost = 9.8

cycle	c0	c1
1	A	
2	B	
3		C
4	D	
5		E
6	F	G
7		
8	H	
9		
10		

cost = 7.6

Spatial and temporal scheduling





cost = 7.6

Approaches

- Superblock instruction scheduling is NP-complete
- Heuristic approaches in all commercial and open-source research compilers
    - greedy list scheduling algorithm coupled with a priority heuristic
- Here: Optimal approach
    - useful when longer compile times are tolerable
    - e.g., compiling for software libraries, digital signal processing, embedded applications, final production build

Outline

- Introduction
    - computer architecture
    - superblock scheduling
- Constraint programming approach
    - temporal scheduler
    - spatial and temporal scheduler
- Experiments
    - experimental setup
    - experimental results
- Lessons learned

Temporal scheduler: Basic constraint model

variables

A, B, C, D, E, F, G

domains

{1, …, *m*}

constraints

B ≥ A + 1, C ≥ A + 1,

D ≥ B + 5, …, G ≥ F

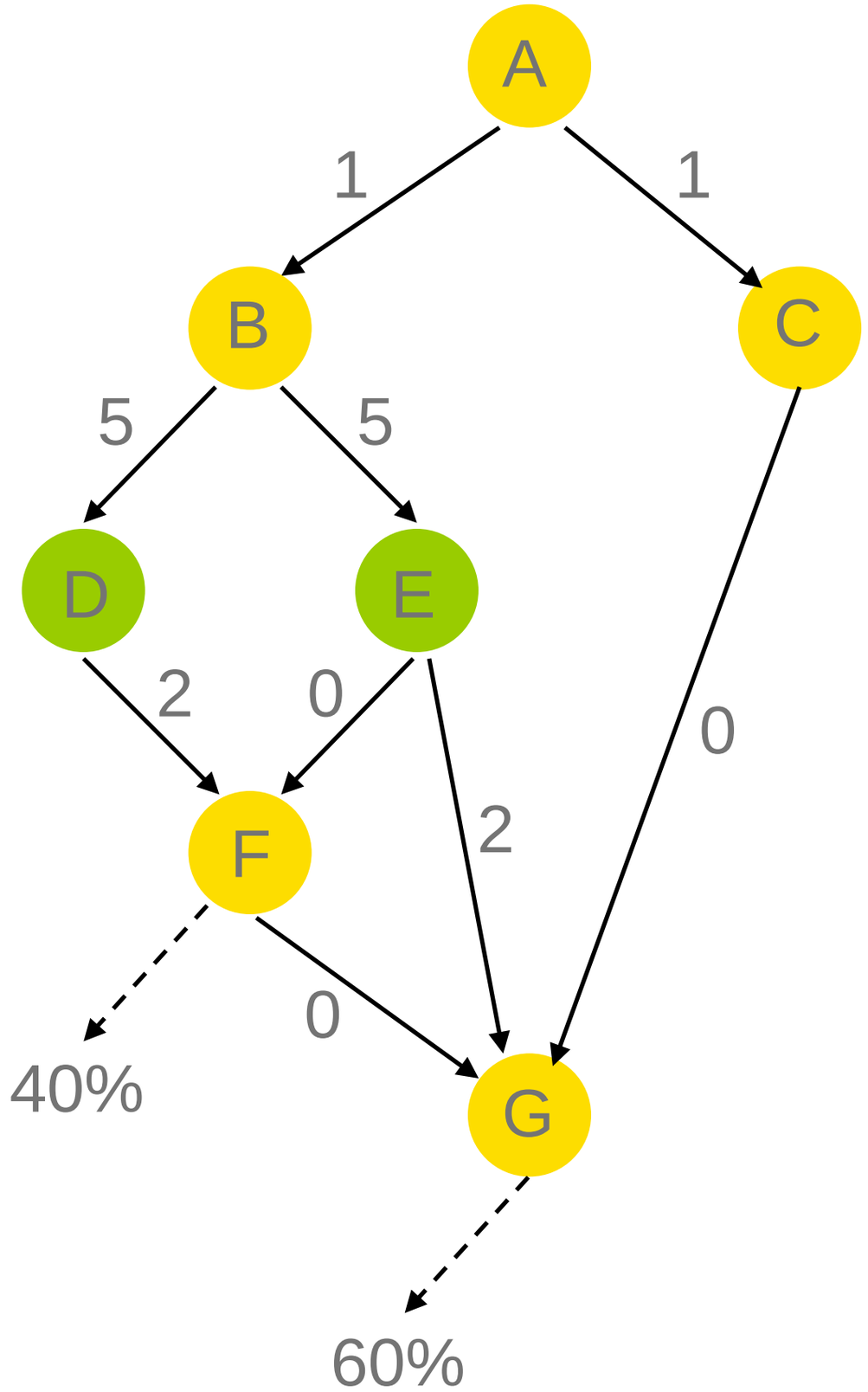
gcc(A, B, C, F, G, *nALU*)

gcc(D, E, *nFPU*)

gcc(A, …, G, *issuewidth*)

cost function

40×F + 60×G



Temporal scheduler
Basic constraint model (con't)

non-fully pipelined instructions

- *introduce auxiliary variables*

  $P_{B,1}$

  $P_{B,2}$

- *introduce additional constraints*

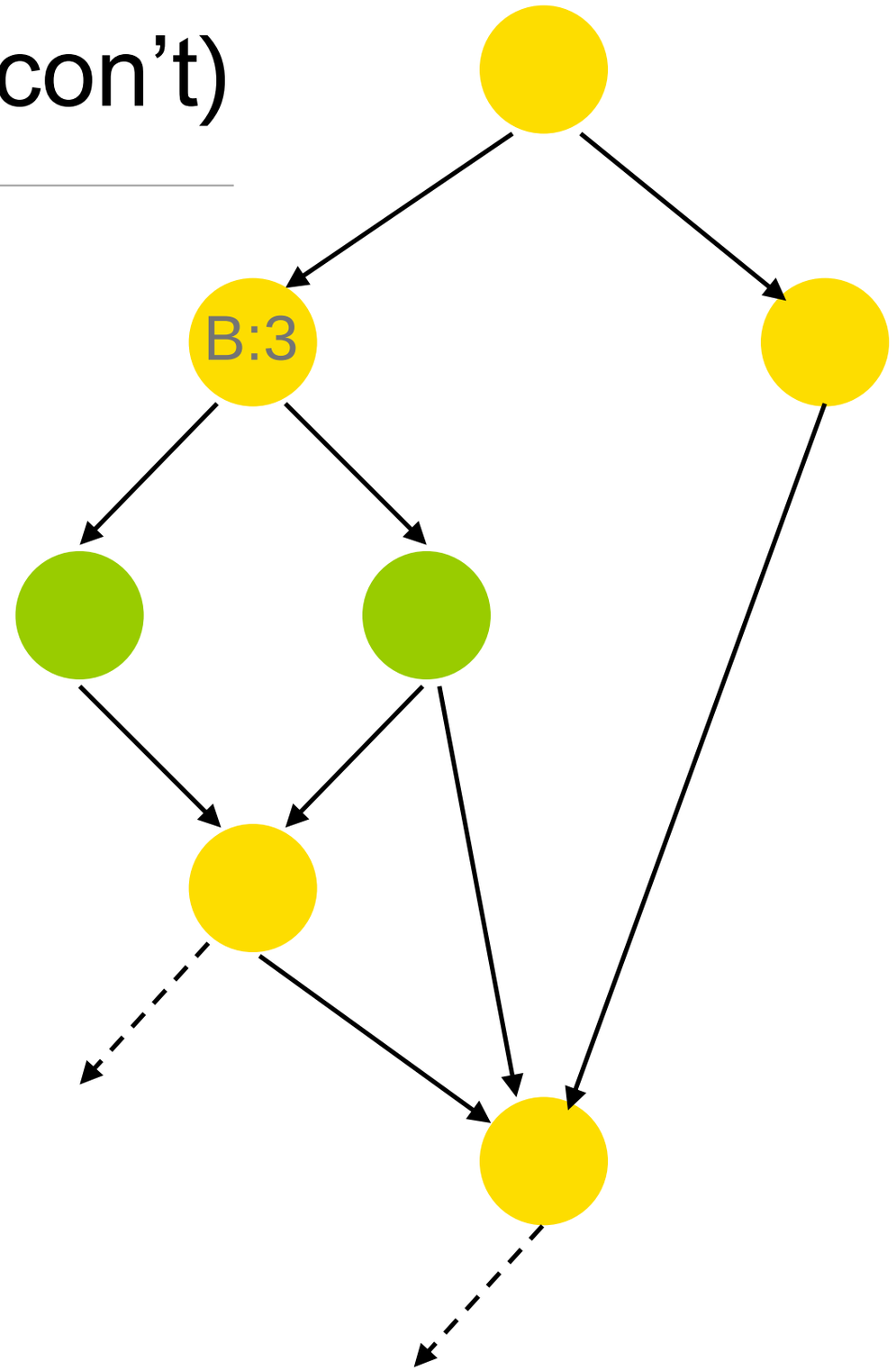
  $B + 1 = P_{B,1}$

  $B + 2 = P_{B,2}$

  gcc(A, B, $P_{B,1}$, $P_{B,2}$ C, F, G, *nALU*)

serializing instructions

- *similar technique*



Temporal scheduler: Improving the model

- Add constraints to increase constraint propagation (e.g., Smith 2006)
    - implied constraints: do not change set of solutions
    - dominance constraints: preserve an optimal solution
- Here:
    - many constraints added to constraint model in extensive preprocessing stage that occurs once
    - extensive preprocessing effort pays off as model is solved many times

Temporal scheduler: Improving the solver

- From optimization to satisfaction
    - find bounds on cost function
    - enumerate solutions to cost function (knapsack constraint; Trick 2001)
    - step through in increasing order of cost
- Improved bounds consistency algorithm for gcc constraints
- Use portfolio to improve performance (Gomes et al. 1997)
    - increasing levels of constraint propagation
- Impact-based variable ordering (Refalo 2004)
- Structure-based decomposition technique (Freuder 1994)

Spatial and temporal scheduler: Basic constraint model

variables

cycle of issue: $x_A, x_B, \ldots, x_H$

cluster: $y_A, y_B, \ldots, y_H$

domains

$\mathrm{dom}(x) = \{1, \ldots, m\}$

$\mathrm{dom}(y) = \{0, \ldots, k-1\}$

communication constraints

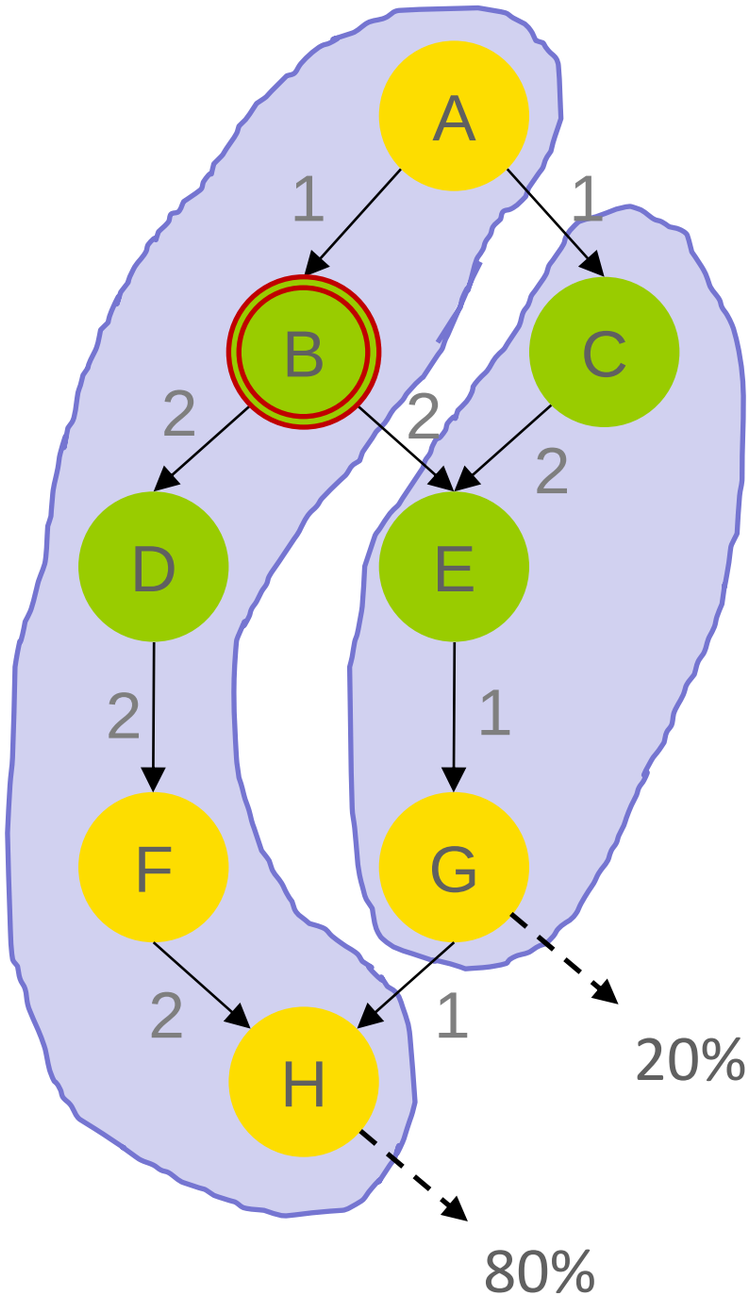
$y_A \neq y_C \rightarrow x_C \geq x_A + 1 + \mathrm{cost}$

$y_A = y_C \rightarrow x_C \geq x_A + 1$

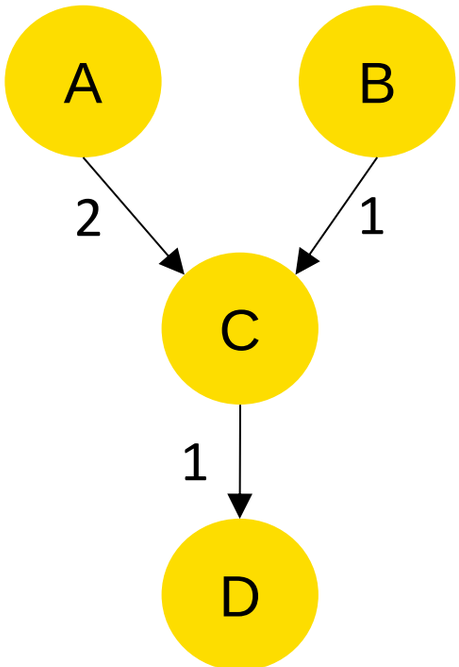
…

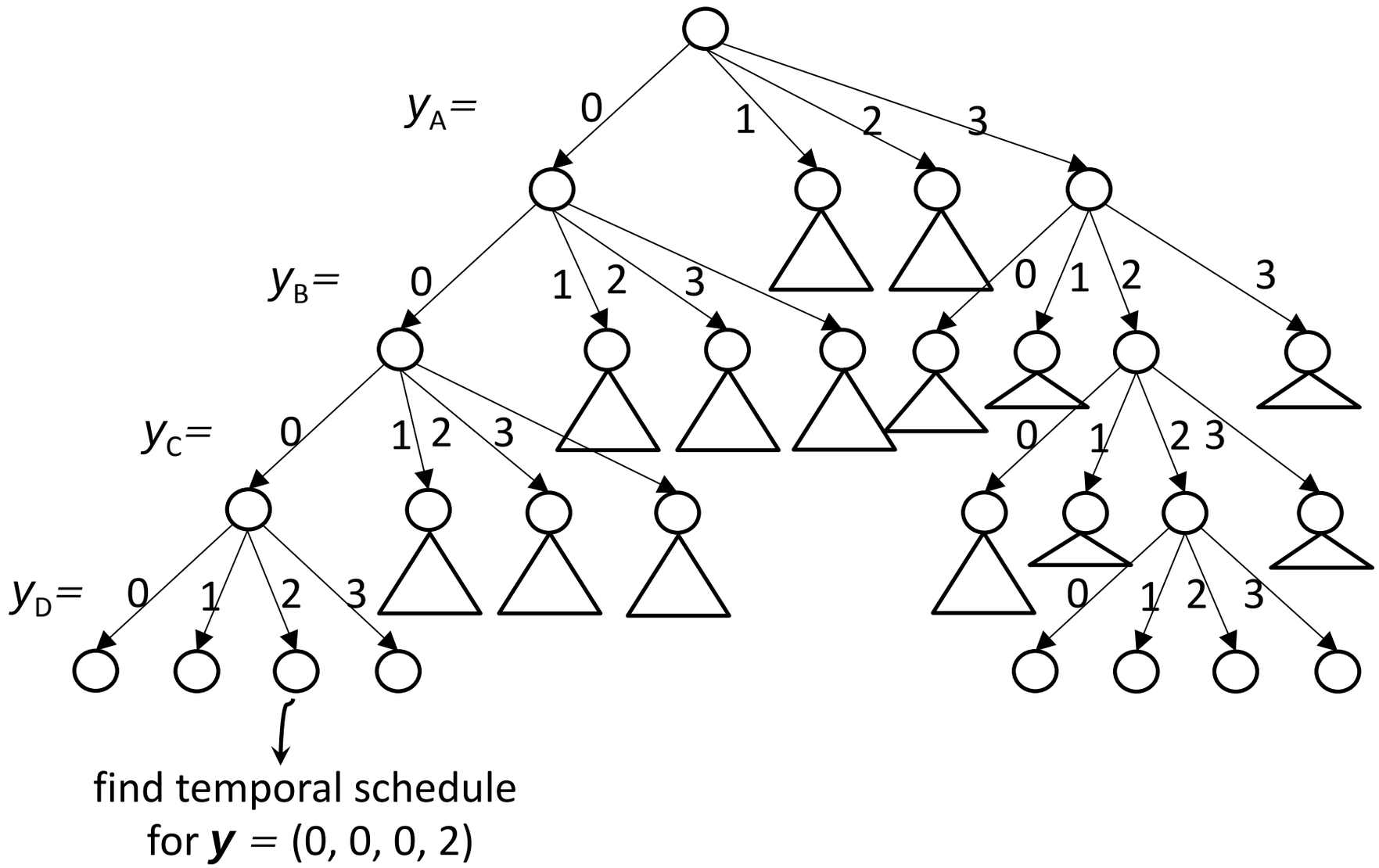
cost function

$80 \times x_H + 20 \times x_G$



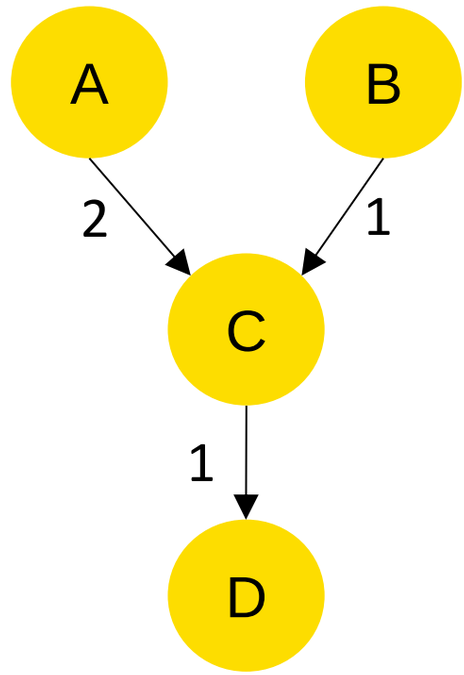
Spatial and temporal scheduler: Search tree of basic model



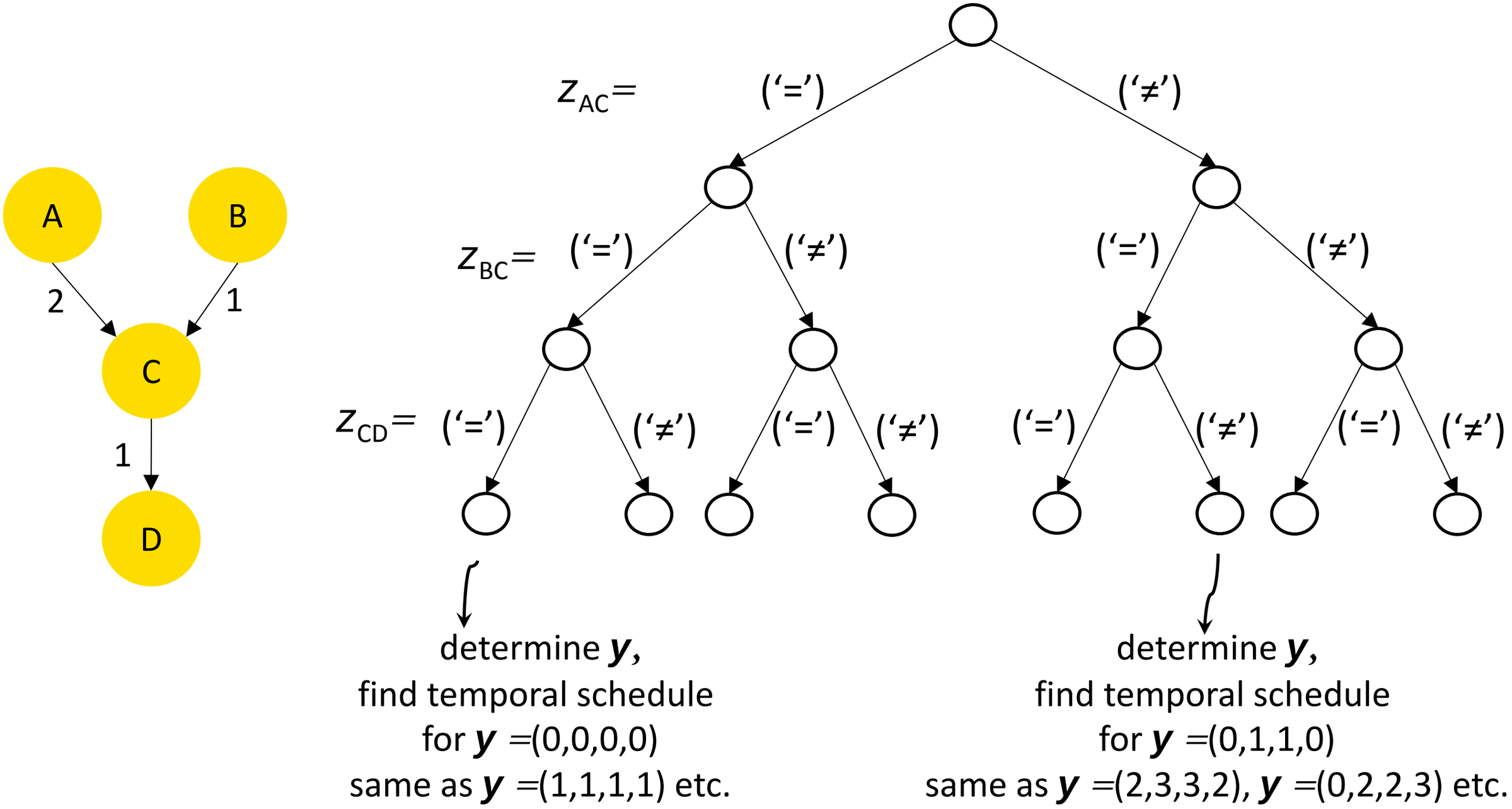


Spatial and temporal scheduler: Improving the model

- Symmetry breaking
  - add auxiliary variables: z_{AC}, z_{BC}, …
  - dom(z) = {‘=’, ‘≠’}
  - instead of backtracking on the y’s backtrack on the edges with z’s
  - preserves at least one optimal solution



Spatial and temporal scheduler: Search tree of improved model



Spatial and temporal scheduler: Improving the solver

- Preprocess DAG to find instructions which must be on same cluster
    - preserve an optimal solution
- Variable ordering
    - assign *z* variables first, in breadth-first order of DAG
    - determine assignment for corresponding *y* variables
    - determine cost of temporal schedule for these assignments

Outline

- Introduction
    - computer architecture
    - superblock scheduling
- Constraint programming approach
    - temporal scheduler
    - spatial and temporal scheduler
- Experiments
    - experimental setup
    - experimental results
- Lessons learned

Experimental setup: Instances

- All 154,651 superblocks from SPEC 2000 integer and floating pt. benchmarks
    - standard benchmark suite
    - consists of software packages chosen to be representative of types of programming languages and applications
    - superblocks generated by IBM’s Tobey compiler when compiling the software packages
    - compilations done using Tobey’s highest level of optimization

Experimental setup: Target architectures

Realistic architectures:

- not fully pipelined
- issue width not equal to number of functional units
- serializing instructions

architecture	issue width	simple int. units	complex int. units	memory units	branch units	floating pt. units
1-issue	1	1				
2-issue	2	1		1	1	1
4-issue	4	2	1	1	1	1
6-issue	6	2		2	3	2

Experimental results: Temporal scheduler

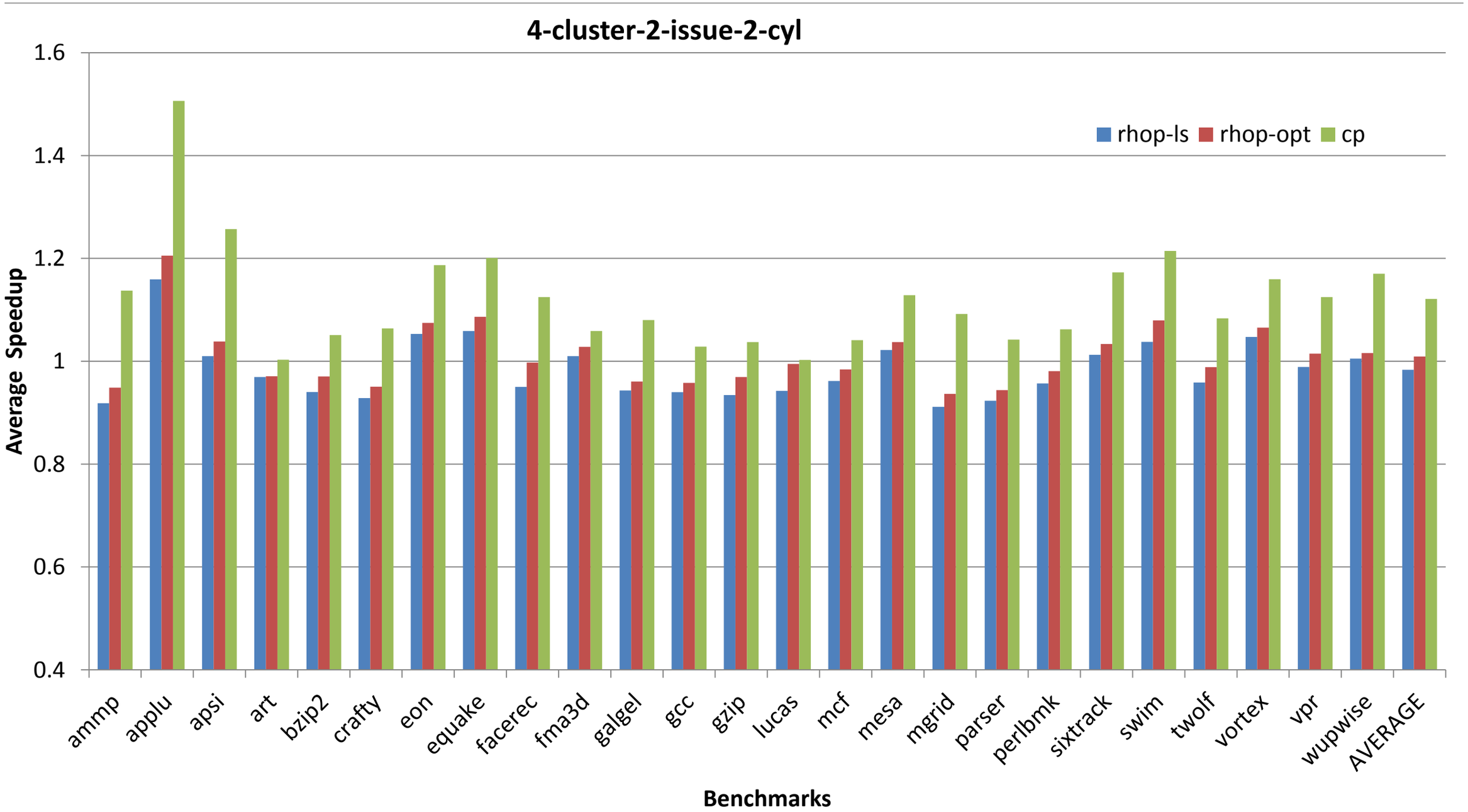
Total time (hh:mm:ss) to schedule *all* superblocks and percentage solved to optimality, for various time limits for solving each instance

	1 sec.		10 sec.		1 min.		10 min.	
architecture	time	%	time	%	time	%	time	%
1-issue	1:30:20	97.34	7:15:46	99.38	10:22:36	99.96	15:08:44	99.98
2-issue	3:57:13	91.83	30:53:83	93.90	108:50:01	97.18	665:31:00	97.70
4-issue	2:17:44	95.47	17:09:48	96.60	61:29:31	98.43	343:04:46	98.87
6-issue	3:04:18	93.59	25:03:44	94.76	87:04:34	97.78	511:19:14	98.29

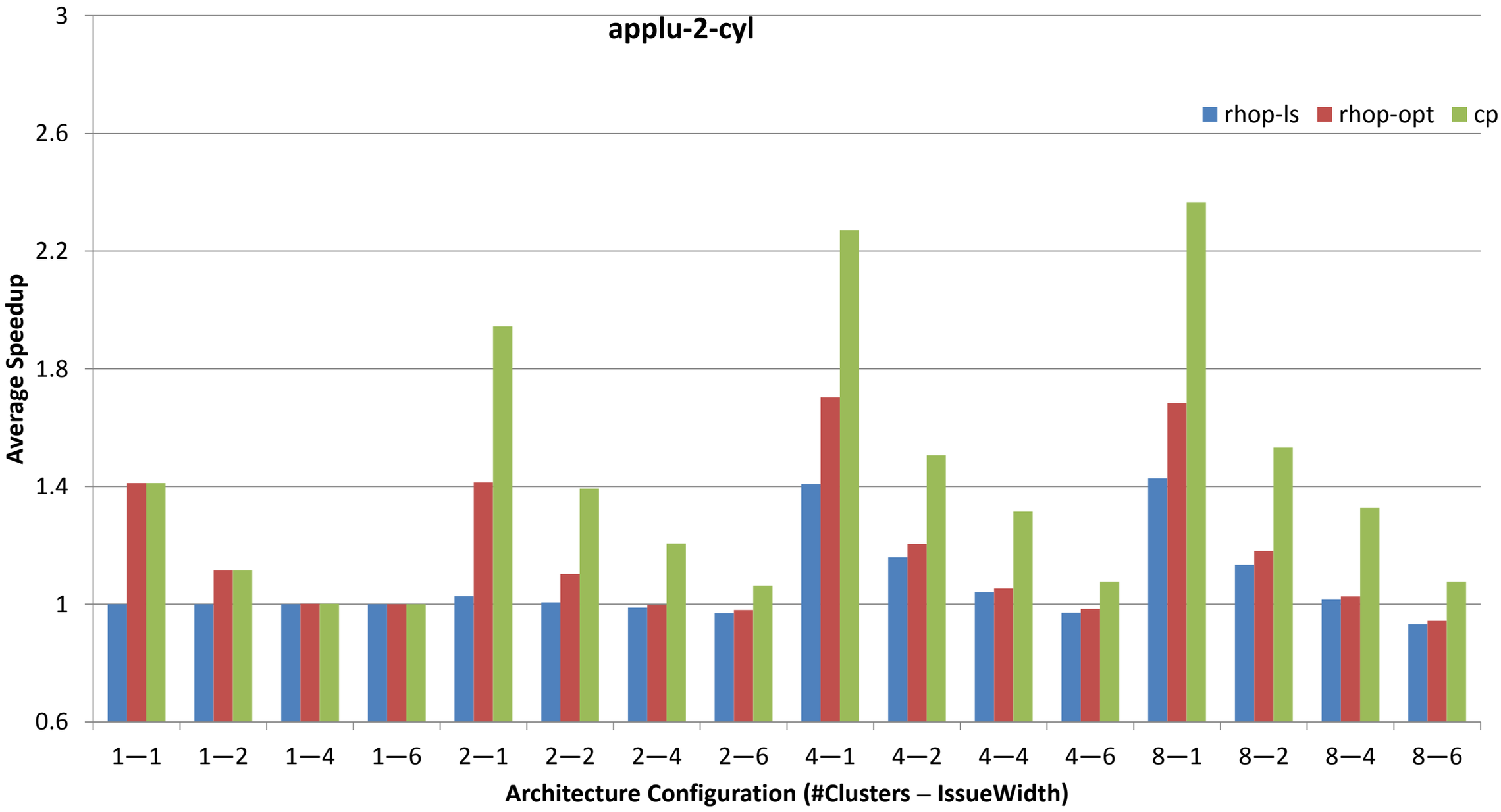
Spatial and temporal scheduler: Some related work

- Bottom Up Greedy (BUG) [Ellis. MIT Press ‘86]
  - greedy heuristic algorithm
  - localized clustering decisions
- Hierarchical Partitioning (RHOP) [Chu et al. PLDI ‘03]
  - coarsening and refinement heuristic
  - weights of nodes and edges updated as algorithm progresses

Experimental results:
Spatial and temporal scheduler



Experimental results:
Spatial and temporal scheduler



Outline

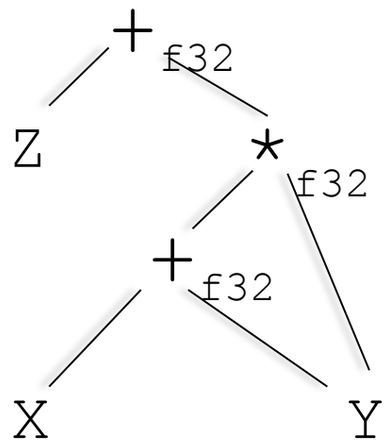
- Introduction
  - computer architecture
  - superblock scheduling
- Constraint programming approach
  - temporal scheduler
  - spatial and temporal scheduler
- Experiments
  - experimental setup
  - experimental results
- Lessons learned

Lessons learned (I)

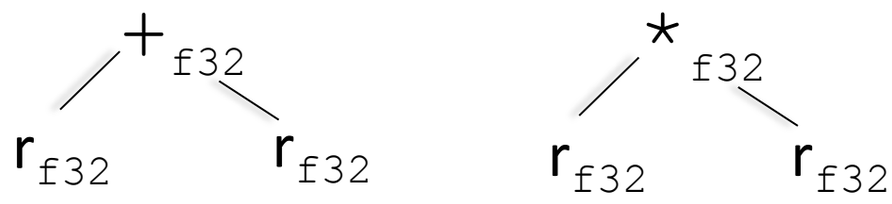
- Pick problem carefully
    - is a new solution needed?
    - what is the likelihood of success?
- Existing heuristics may not leave any room for improvement

Instruction Selection

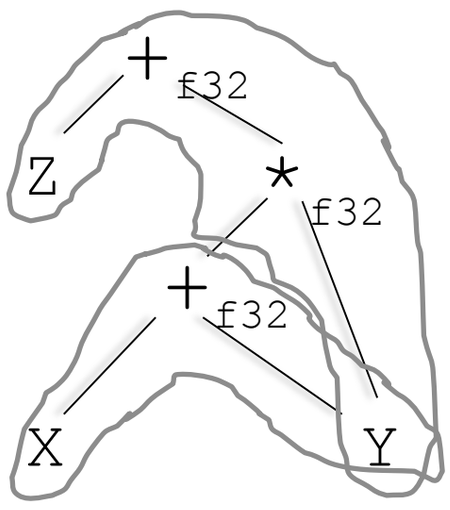
DAG:



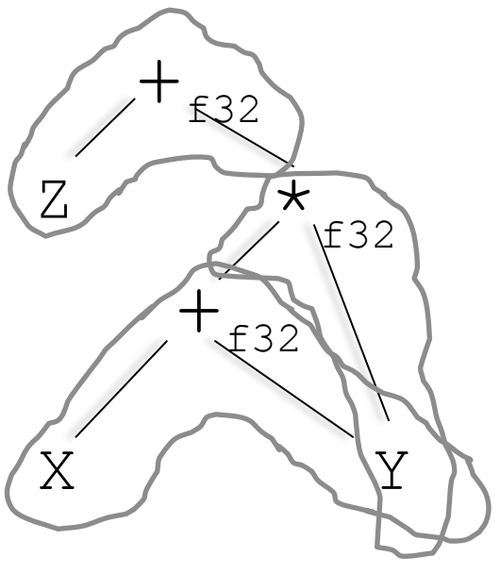
TILES:



OUTPUT:



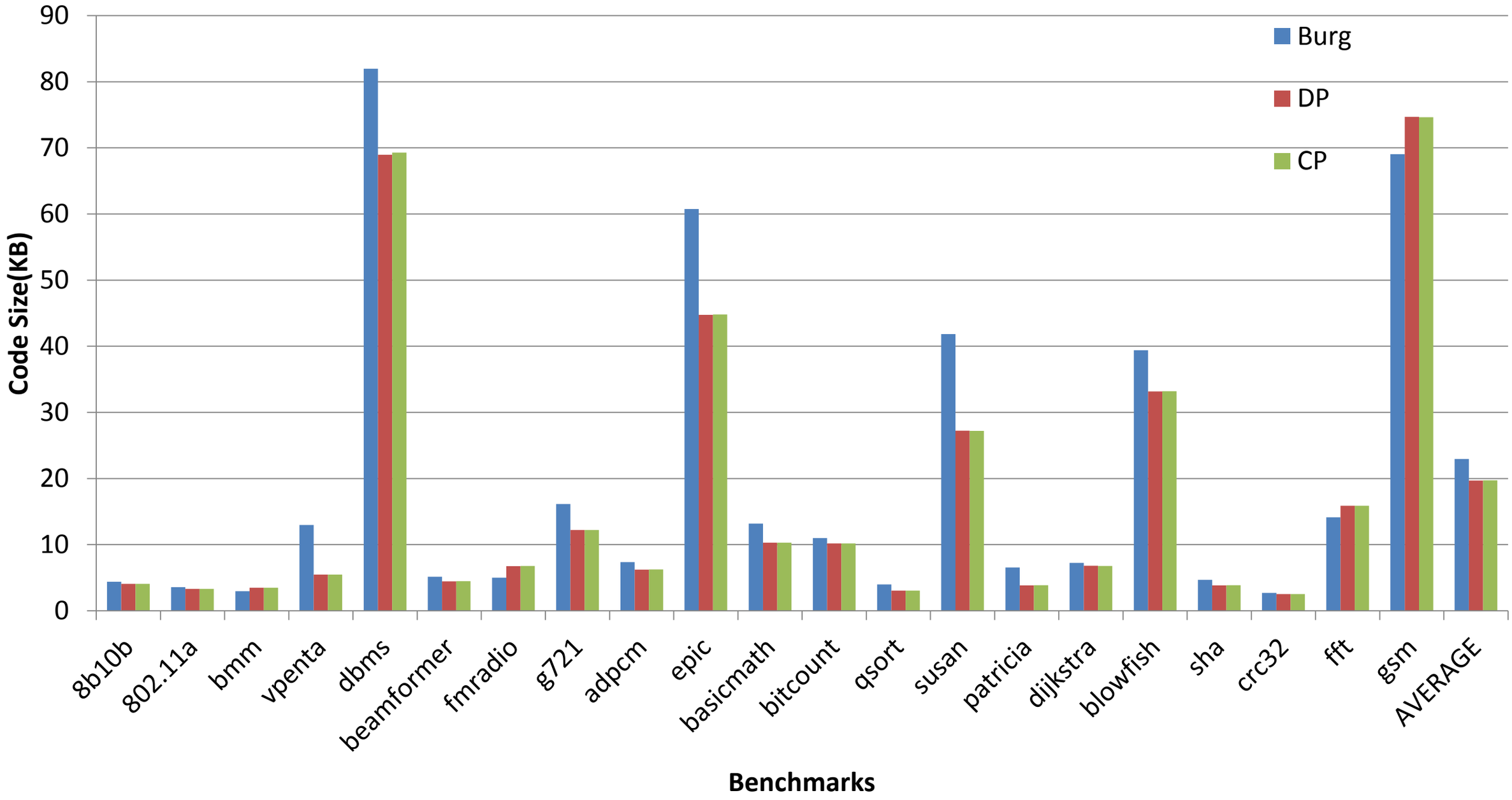
OR



Instruction Selection

- Given
    - an expression DAG ***G***
    - a set of tiles representing machine instructions
- Find a mapping of tiles to nodes in ***G*** of minimal cost (size) that covers ***G***
- Complexity:
    - polynomial for trees
    - NP-hard for DAGs

Experimental evaluation



Lessons learned (II)

- Be prepared for adversity
    - significant overhead
        - learning domain of application
        - significant implementation
        - significant engineering
- different research cultures
    - researchers are tribal
    - different standards of reviewing (number & contentiousness)
    - different standards of evaluation, formalization, assumptions

Lessons learned (III)

- Rewards
    - can be attractive to students
    - can lead to identifying and solving interesting sub-problems whose solutions have general applicability
    - bounds consistency for alldifferent and gcc global constraints
    - restarts and portfolios
    - machine learning of heuristics

Optimization problems in compilers

- Instruction selection
- Instruction scheduling
    - basic-block scheduling
    - super-block scheduling
    - loop scheduling: tiling, unrolling, fusion
- Memory hierarchy optimizations
- Register allocation

Selected publications

- Applications

  M. Beg and P. van Beek. A constraint programming approach for integrated spatial and temporal scheduling for clustered architectures. *ACM TECS*, 2013.

  A. M. Malik, M. Chase, T. Russell, and P. van Beek. An application of constraint programming to superblock instruction scheduling. *CP-2008*.

- Global constraints

  C.-G. Quimper, P. van Beek, A. Lopez-Ortiz, A. Golynski, and S. Bashir Sadjad. An efficient bounds consistency algorithm for the global cardinality constraint. *CP-2003*.

  A. Lopez-Ortiz, C.-G. Quimper, J. Tromp, and P. van Beek. A fast and simple algorithm for bounds consistency of the alldifferent constraint. *IJCAI-2003*.

- Portfolios and restarts

  H. Wu and P. van Beek. On portfolios for backtracking search in the presence of deadlines. *ICTAI-2007*.

  H. Wu and P. van Beek. On universal restart strategies for backtracking search. *CP-2007*.

- Heuristics and machine learning

  M. Chase, A. M. Malik, T. Russell, R. W. Oldford, and P. van Beek. A computational study of heuristic and exact techniques for superblock instruction scheduling. *J. of Scheduling*, 2012.

  T. Russell, A. M. Malik, M. Chase, and P. van Beek. Learning heuristics for the superblock instruction scheduling problem. *IEEE TKDE*, 2009.

Next project:
Smart water infrastructure