Software Pipelining with Iteration Preselection

David Gregg

Institut fuer Computersprachen
Technische Universitaet Wien

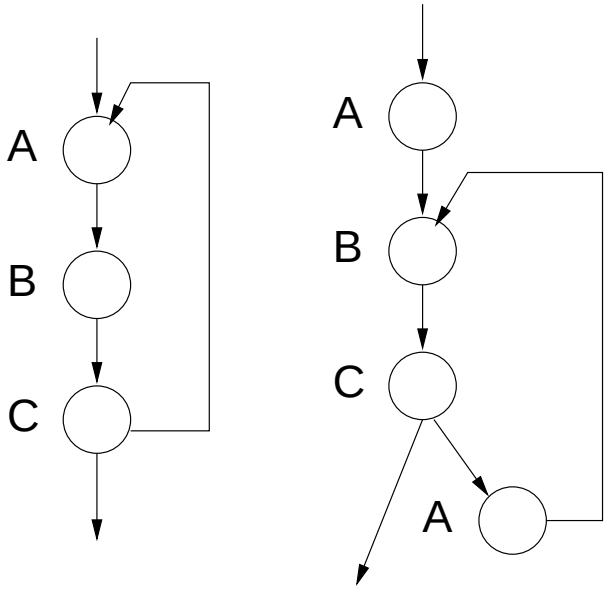
dave@complang.tuwien.ac.at

Software Pipelining Integer Code

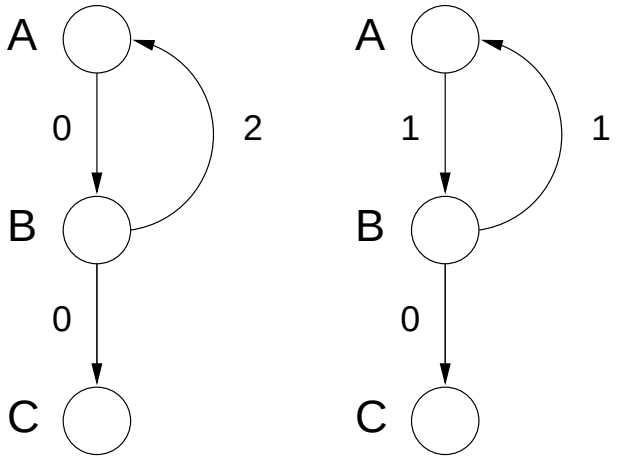
- Frequent poorly predictable branches
  - Must handle loops with branches
  - Variable II - not restricted to longest path
- Low loop trip counts
  - Size of loop prolog is important
  - Avoid unrolling
- Low levels of ILP
  - Dependences are more limiting than resources

Loop Shifting

Pipeline by moving ops above loop entry



CFG



DDG

+1 outgoing
-1 incoming

Extends naturally to loops with branches

Existing Shifting Algorithms

Acyclic Scheduling

- Compact Loop body with acyclic scheduling
- Shift those ops near start of acyclic schedule

Greedy shifting

- No final destination intended

Ad hoc limits on greediness

- Limited scheduling window (EPS)
- Lower iteration number => higher priority (EPS)
- Shift limits (RDLP)

Problems with shifting too much

- Ops become more speculative
  - in loops containing branches
- Register lifetimes longer
- More renaming needed
- Code growth
  - Larger loop prolog
  - More compensation code within loop
- Can increase acyclic dependence length

Decomposed pipelining process

Iteration preselection

- Compute times to shift (TTS) for each op
- Shift ops to correct iteration
- Ignore most resource constraints

Acyclic scheduling

- Schedule resulting loop kernel with acyclic algorithm
- Not confused by pipelining concerns
- Full resource constraints

Iteration Preselection - single path

Calculate approximate MII

Lam’s method

Calculate times to shift (TTS)

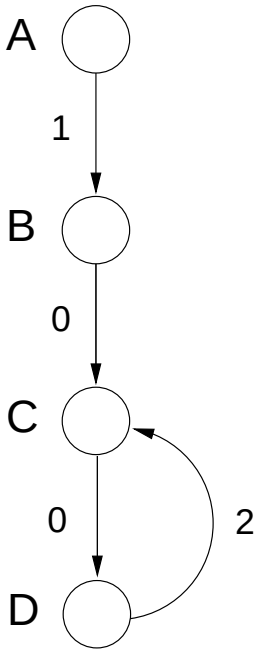
$$TTS = \frac{\text{Circular Height} - 1}{\text{MII}}$$

Shift operations

Code motion rules create loop prolog

Circular Height

DDG



	Height	Circular Height
A	1	3
B	3	3
C	2	2
D	1	1

Repeat until fixed point

Circular height(op) =

latency(op)

+ height(succ)

- weight(op, succ) * MII

MII = 1

All latencies = 1

Multiple Path Loops

Treat each path separately

- Compute TTS for every operation for every path

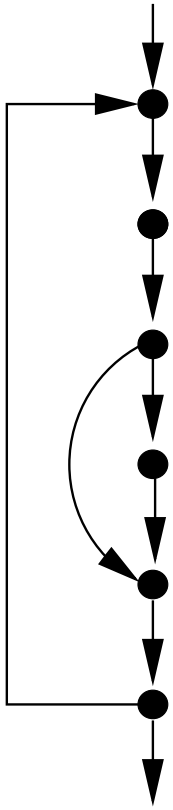
Operations on more than one path

- Choose maximum TTS from all paths the op is on
- If op is duplicated, compensation copy can move independently

Branches

- Circular height comes from op that computes condition

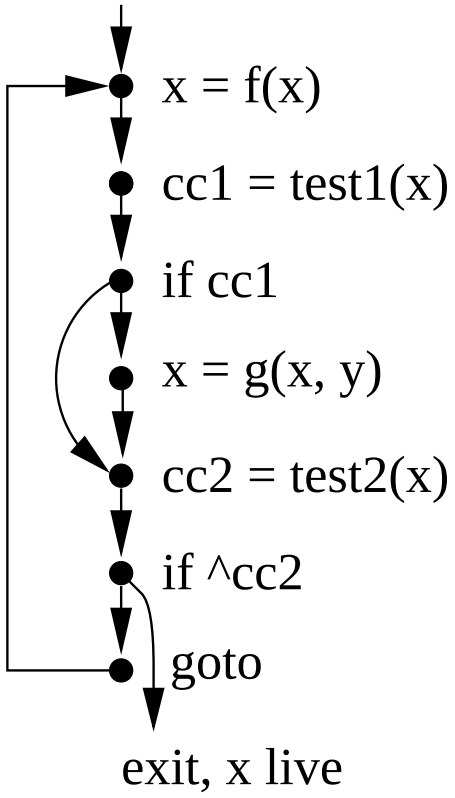
An Example

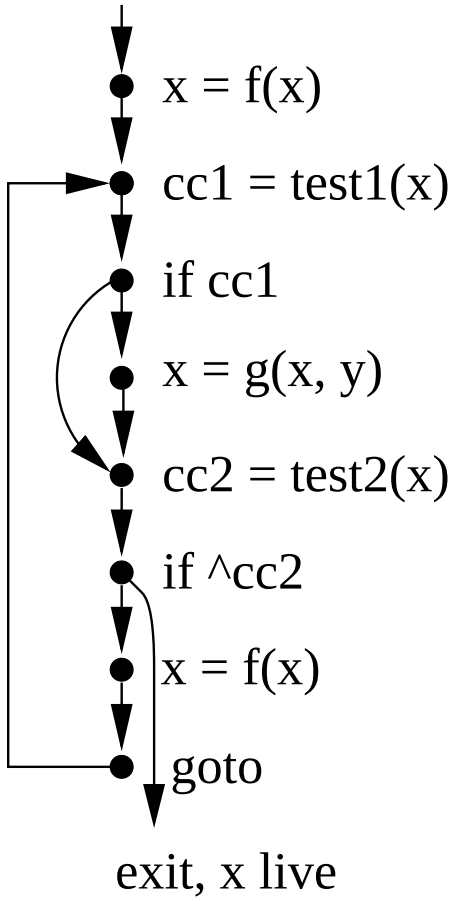


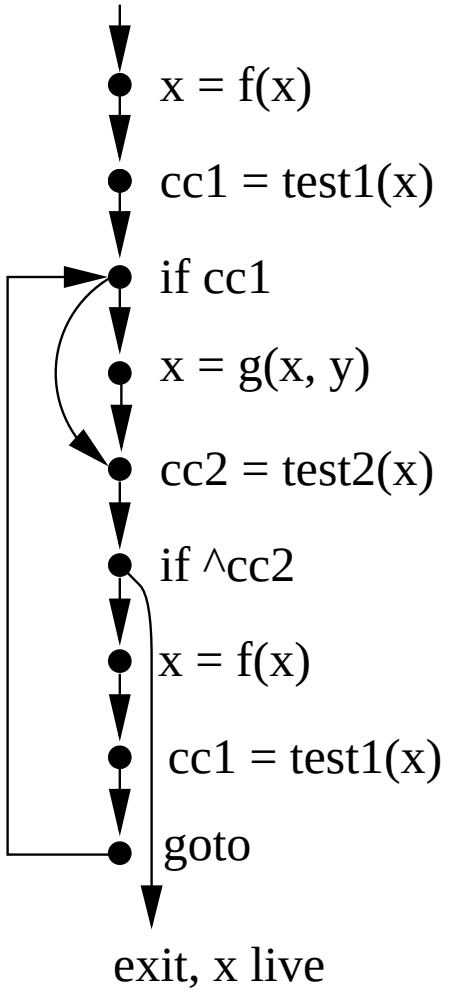
	Path1 (II = 1)		Path2 (II = 2)	
	Circular Height	Times to Shift	Circular Height	Times to Shift
x = f(x)	3	2	4	1
cc1 = test1(x)	2	1	2	0
if cc1	1	0	1	0
x = g(x, y)	-	-	3	1
cc2 = test2(x)	2	1	2	0
if cc2	1	0	1	0

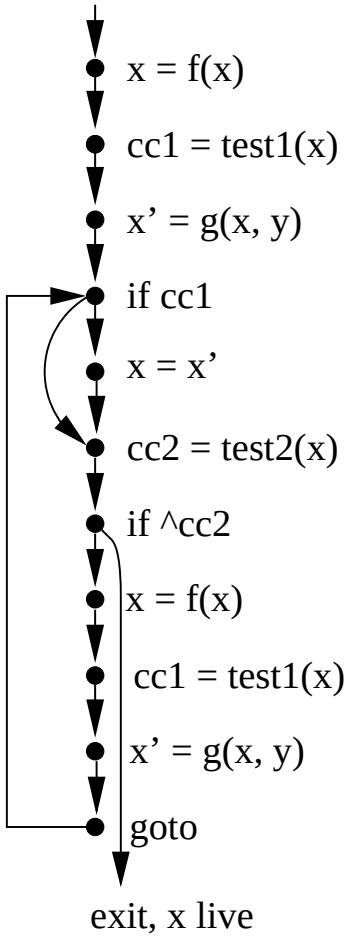
exit, x live

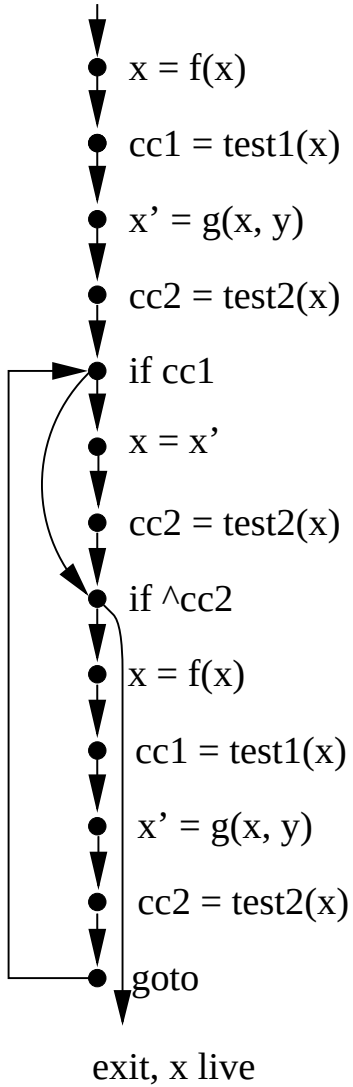
All latencies = 1

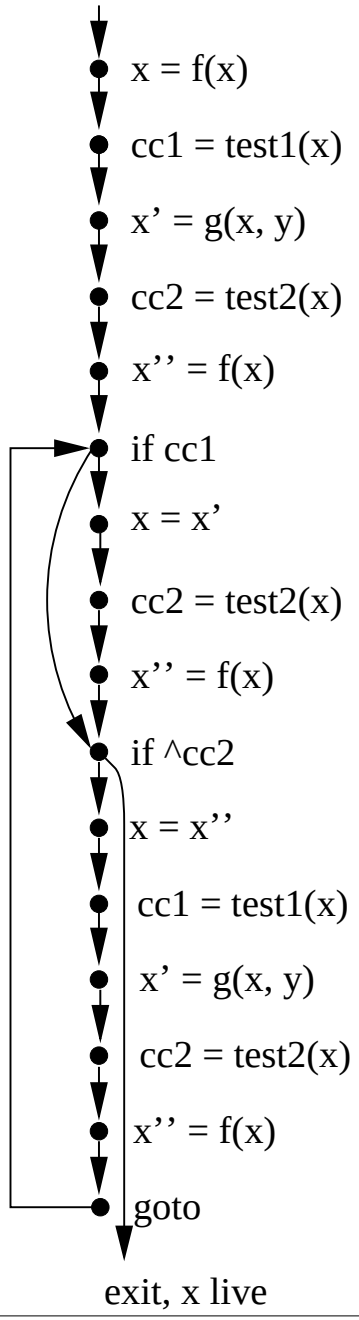

x = f(x)
cc1 = test1(x)
if cc1
x = g(x, y)
cc2 = test2(x)
if ^cc2
goto
exit, x live


x = f(x)
cc1 = test1(x)
if cc1
x = g(x, y)
cc2 = test2(x)
if ^cc2
x = f(x)
goto
exit, x live

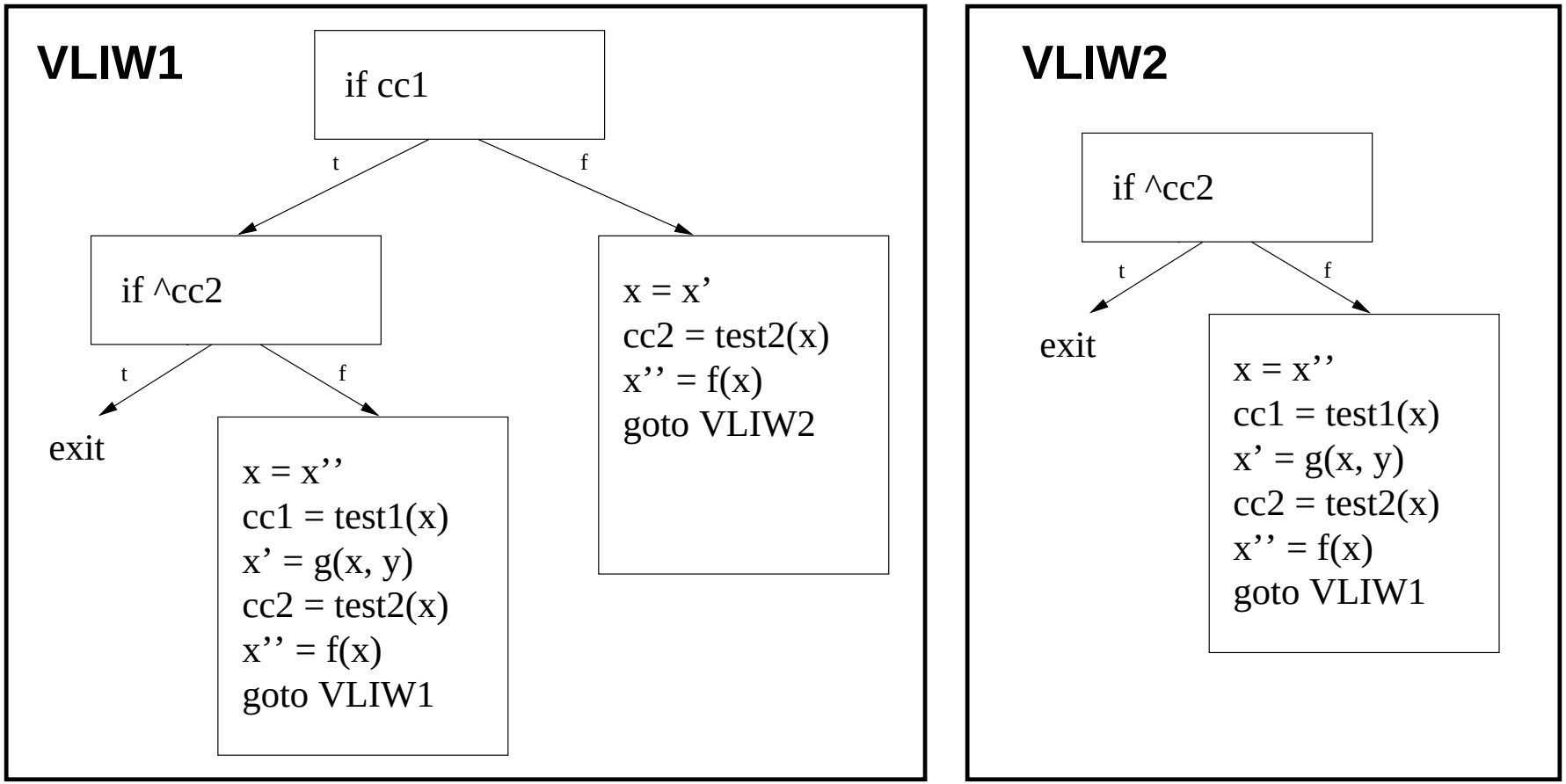

x = f(x)
cc1 = test1(x)
if cc1
x = g(x, y)
cc2 = test2(x)
if ^cc2
x = f(x)
cc1 = test1(x)
goto
exit, x live


x = f(x)
cc1 = test1(x)
x’ = g(x, y)
if cc1
x = x’
cc2 = test2(x)
if ^cc2
x = f(x)
cc1 = test1(x)
x’ = g(x, y)
goto
exit, x live


x = f(x)
cc1 = test1(x)
x’ = g(x, y)
cc2 = test2(x)
if cc1
x = x’
cc2 = test2(x)
if ^cc2
x = f(x)
cc1 = test1(x)
x’ = g(x, y)
cc2 = test2(x)
goto
exit, x live


x = f(x)
cc1 = test1(x)
x’ = g(x, y)
cc2 = test2(x)
x’’ = f(x)
if cc1
x = x’
cc2 = test2(x)
x’’ = f(x)
if ^cc2
x = x’’
cc1 = test1(x)
x’ = g(x, y)
cc2 = test2(x)
x’’ = f(x)
goto
exit, x live

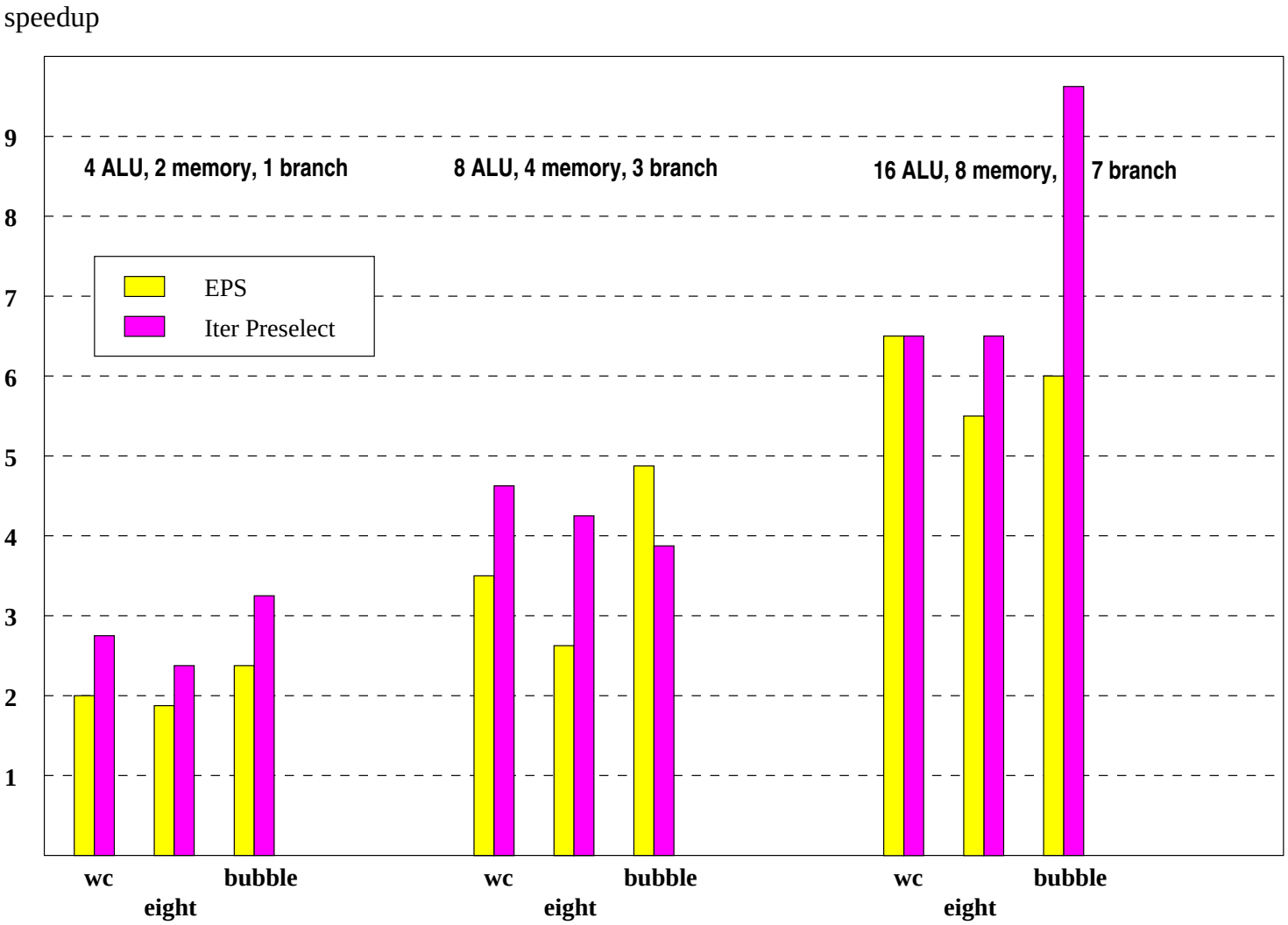
Final Kernel Schedule



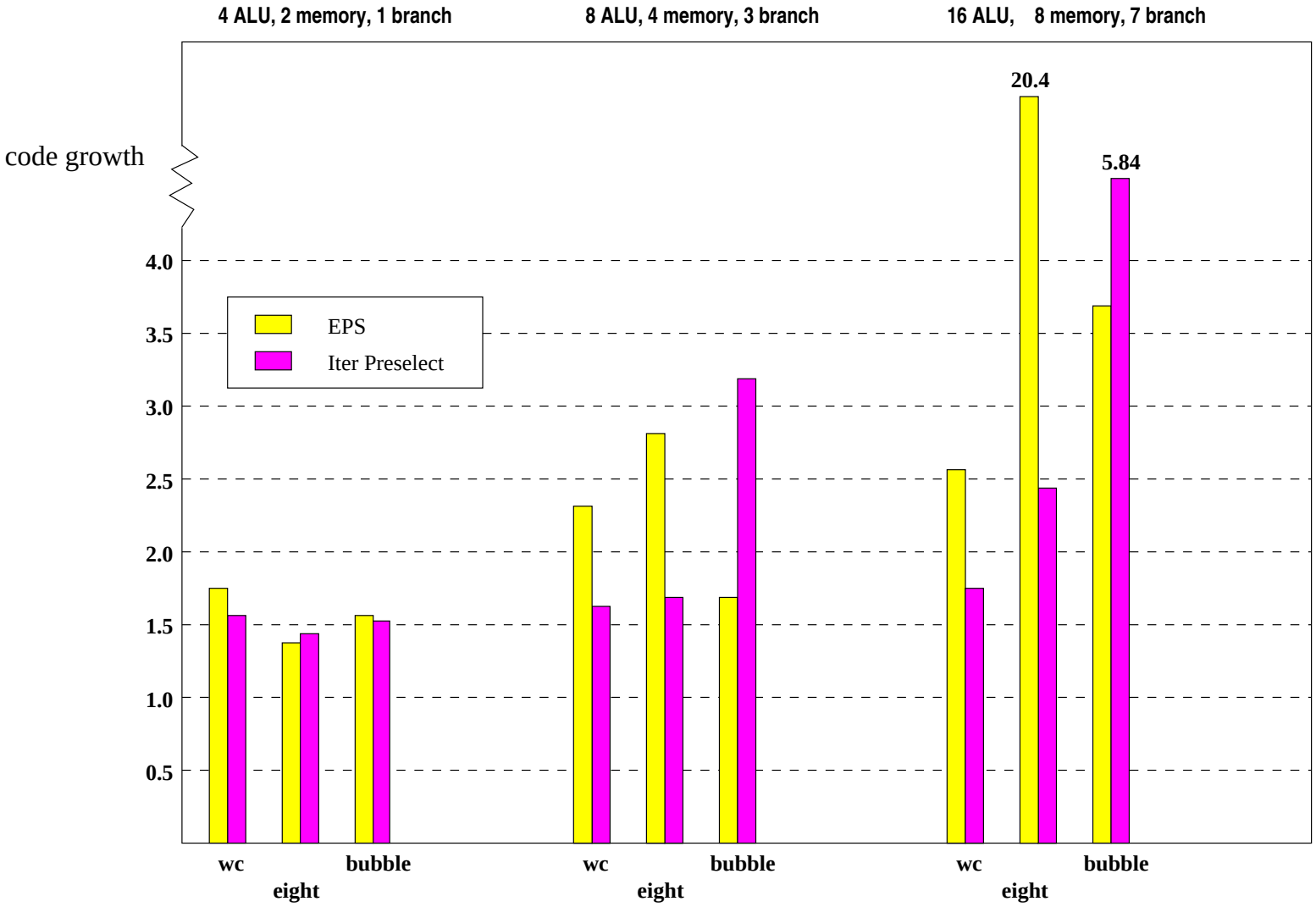
Experimental Results

- Chameleon compiler and simulator
  - Optimising VLIW compiler
  - Simulator for IBM Tree VLIW
- Iteration Preselection
- Enhanced Pipeline Scheduling (EPS)
  - Window size = 16

Results - Speedup



Results - Code Growth



Conclusions, Future Work

- Exponentiality of all paths
  - Only look at K most important paths
  - General height and MII calculation
- Conflicting TTS numbers
  - Find TTS satisfying all paths
  - Priority for more frequent paths
- Theoretical basis
  - Circuit retiming?