Critical Path Reduction for Scalar Programs

Mike Schlansker & Vinod Kathail

Hewlett-Packard Laboratories

HEWLETT PACKARD

What Problem is Addressed?

Generate high performance code for processors with substantial ILP

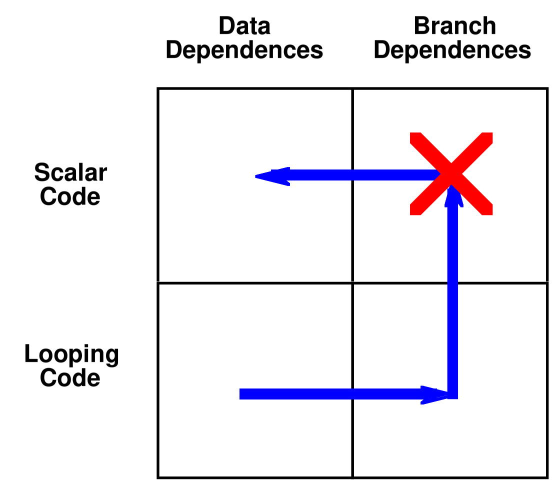
Dependences limit performance with increasing ILP (width × latency)

Alleviate dependence bottlenecks with software rather than hardware

Problem addressed using Critical Path Reduction (CPR)

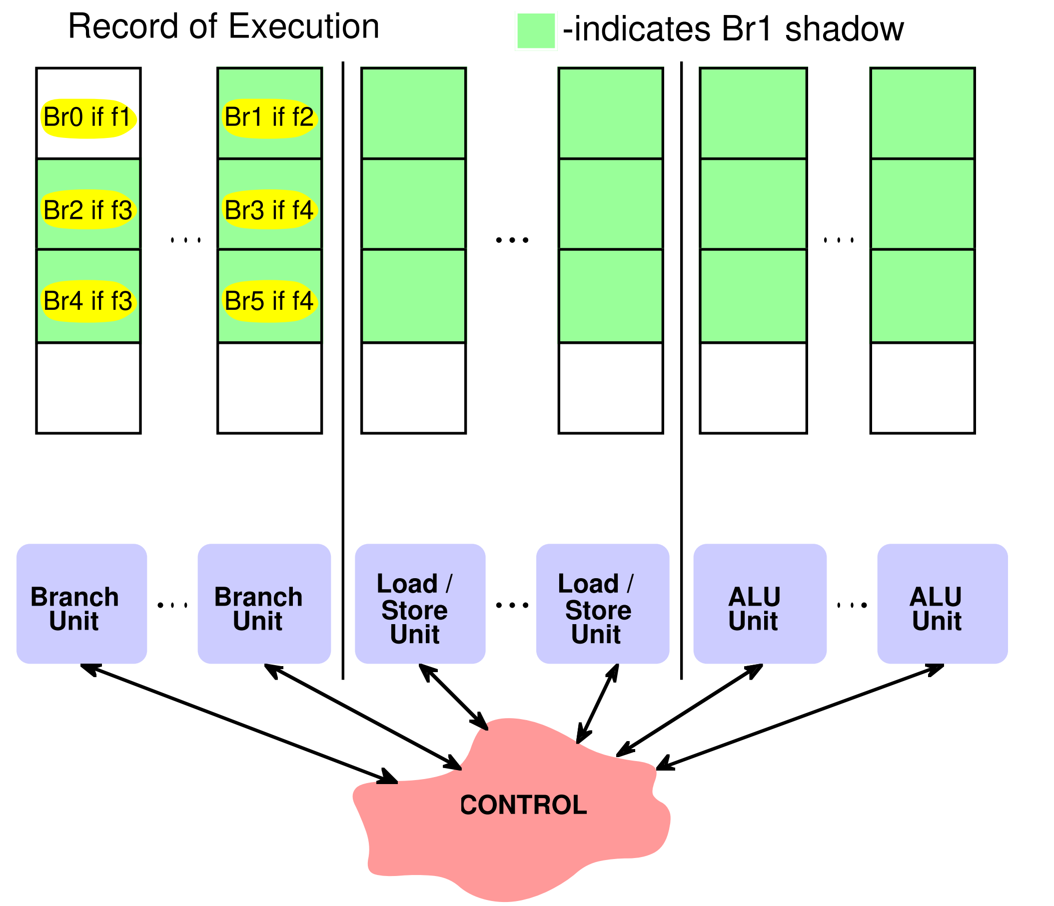
Need a comprehensive theory of CPR
- for data & control
- for loops & scalar

Importance of CPR depends upon:
- amount of hardware parallelism
- application characteristics

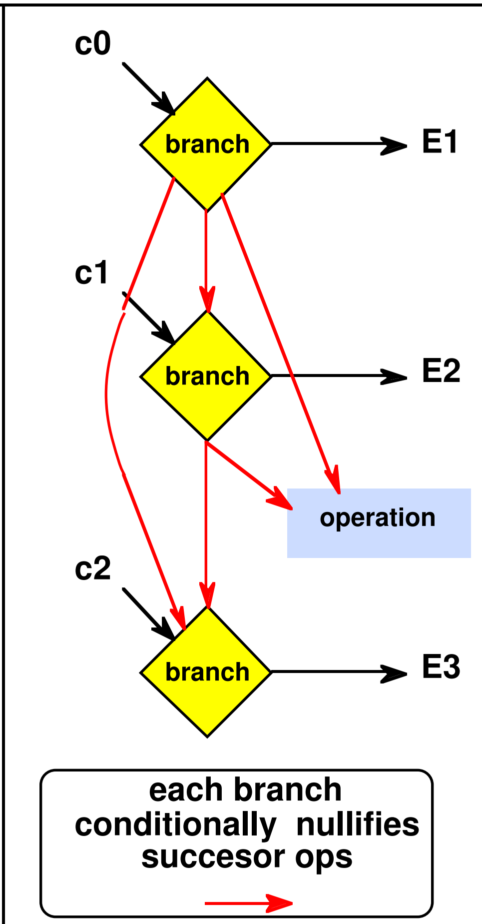


Key CPR principles are re-used in many engineering settings

Simplify Hardware for Enforcing Branch Dependences



Substantial ILP means geographically distributed Hardware



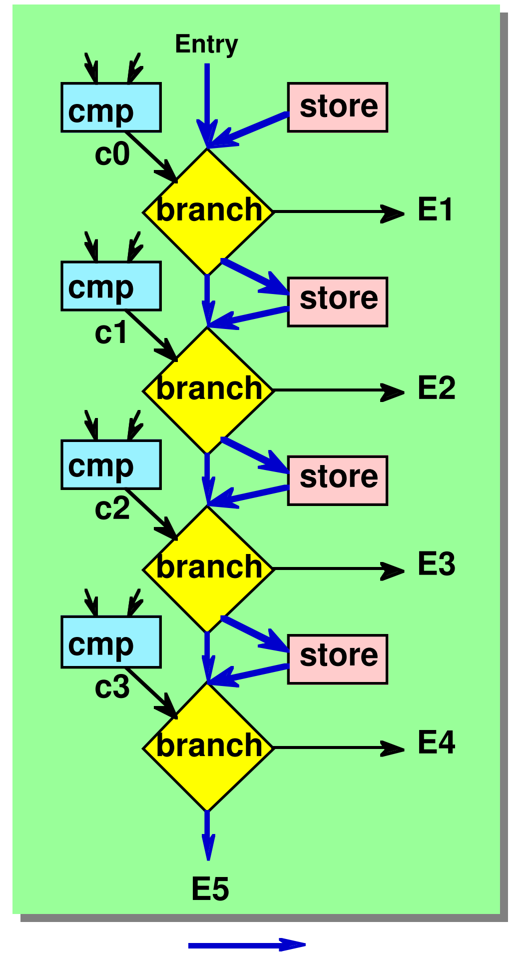
Hardware must enforce branch dependences

Alleviate rigid constraints in superblock scheduling

Superblock and hyperblock scheduling
- currently in use within our compiler

Benefits of superblock scheduling:
- avoids complexity of program merges
- local scheduling technology
  - no compensation code
  - no compensation blocks

Problems with traditional superblock scheduling:
- excessive schedule length
- branches are sequential & trapped on trace
- non-speculative ops trapped between branches



Branch dependences within a superblock

HEWLETT PACKARD

Prior Work

Data height reduction
- Tree height reduction: Kuck
- Multiflow compiler: Lowney et. al. 1993
- Cydrome compiler: Dehnert et. al. 1993
- Data recurrences on vector processors: Y. Tenaka et. al. 1988
- Data recurrences on ILP processors: Schlansker & Kathail 1993

Control height reduction
- Speculation: Fisher, Ellis, Nicolau, Ebcioglu, Hwu, Tirumala
- Exchanging branch order: Fisher, Ellis, Nicolau, Moon & Ebcioglu
- Hardware for concurrent branch:
  - Fisher 1980 - Multiflow Trace architecture;
  - Moon 1992
- Control recurrences in loops: - Schlansker & Kathail & Anik 1994

Predictability of program traces - Fisher & Freudenberger 92

Results

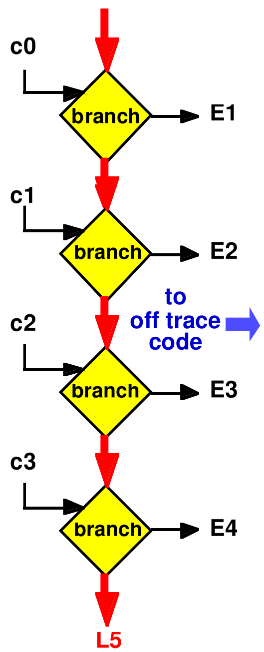
CPR techniques have been developed for:
- software pipeline loops (prior work)
- superblock scheduling (here)
- predicated execution (HPL PlayDoh)
- joint treatment of control and data dependences

Results are in two forms:
- Code transformations to reduce critical path length
- Scheduling techniques compatible with CPR

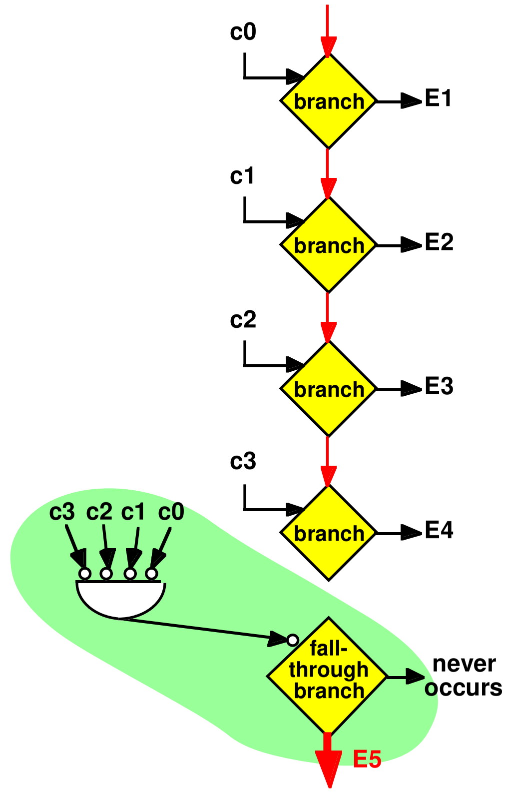
CPR transformations are presented which:
- Minimize dependence height
- Minimize redundant computation

A scheduling approach is presented which:
- Takes full advantage of CPR
- Preserves the simplicity of superblock scheduling
- Provides a simple approach for scheduling compensation code

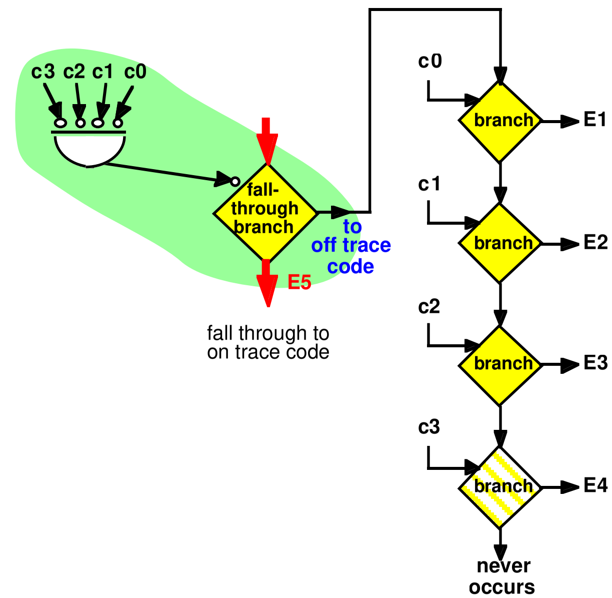
Basic Idea - Control CPR using branches



Superblock CFG



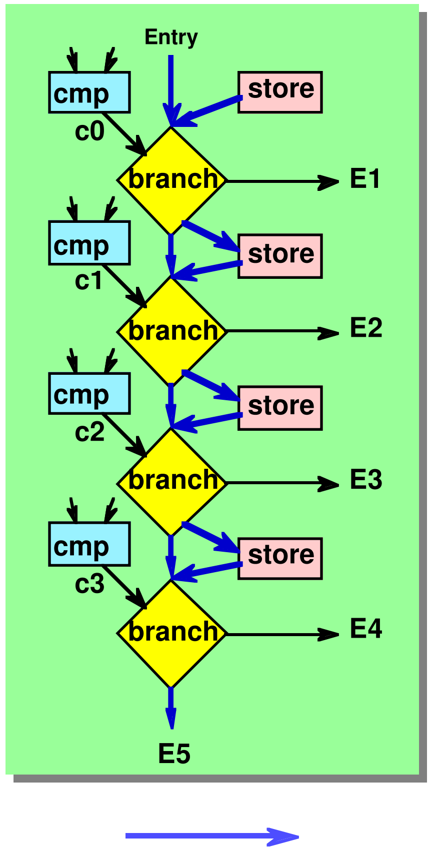
Insert lookahead Branch



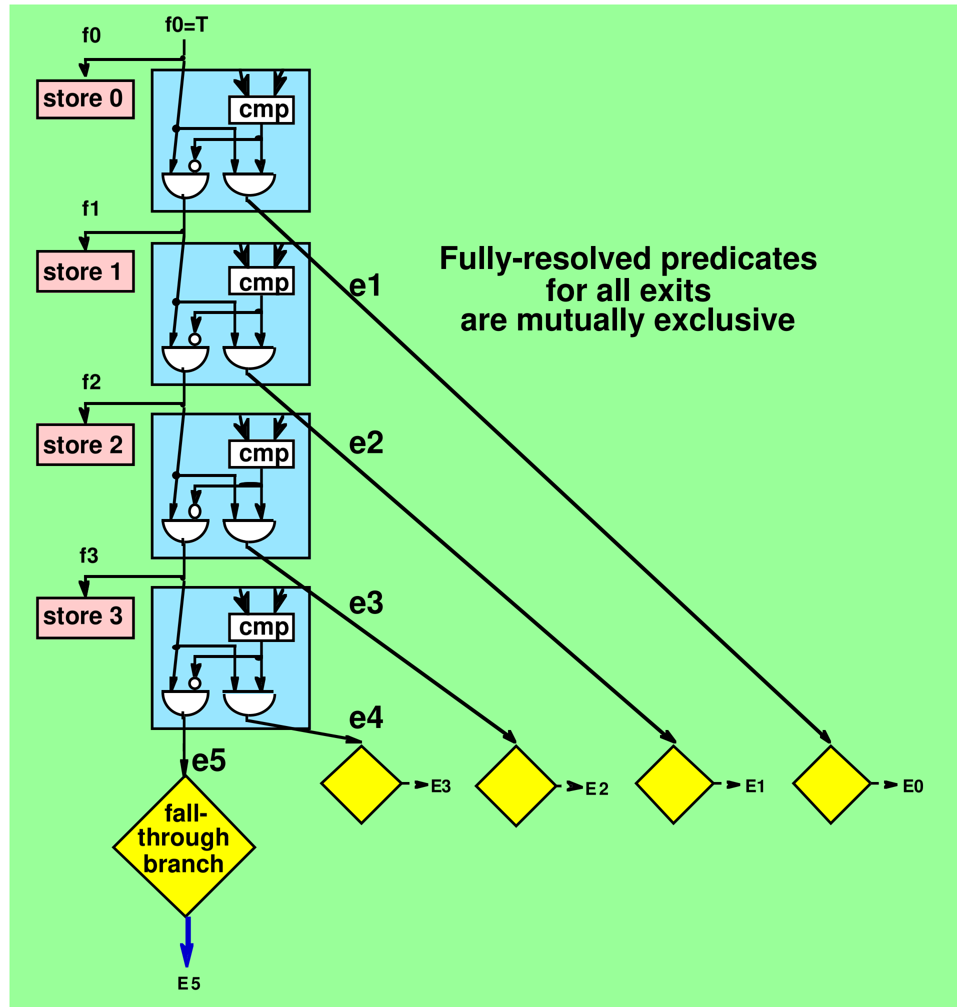
Move non-lookahead branches off-trace

HEWLETT PACKARD

Control CPR using Fully-Resolved Predicates



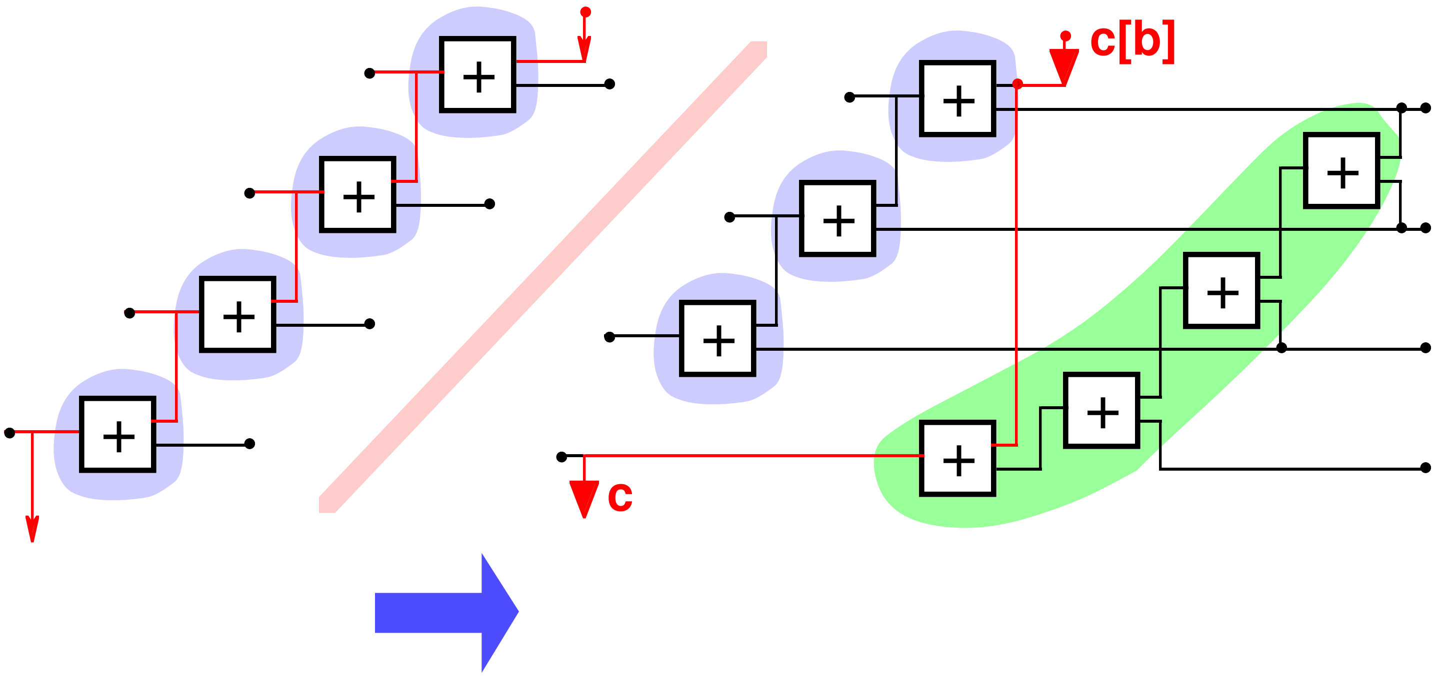
Branch dependences in a superblock



FRPs - branch dependence $\Rightarrow$ data dependence

HEWLETT PACKARD

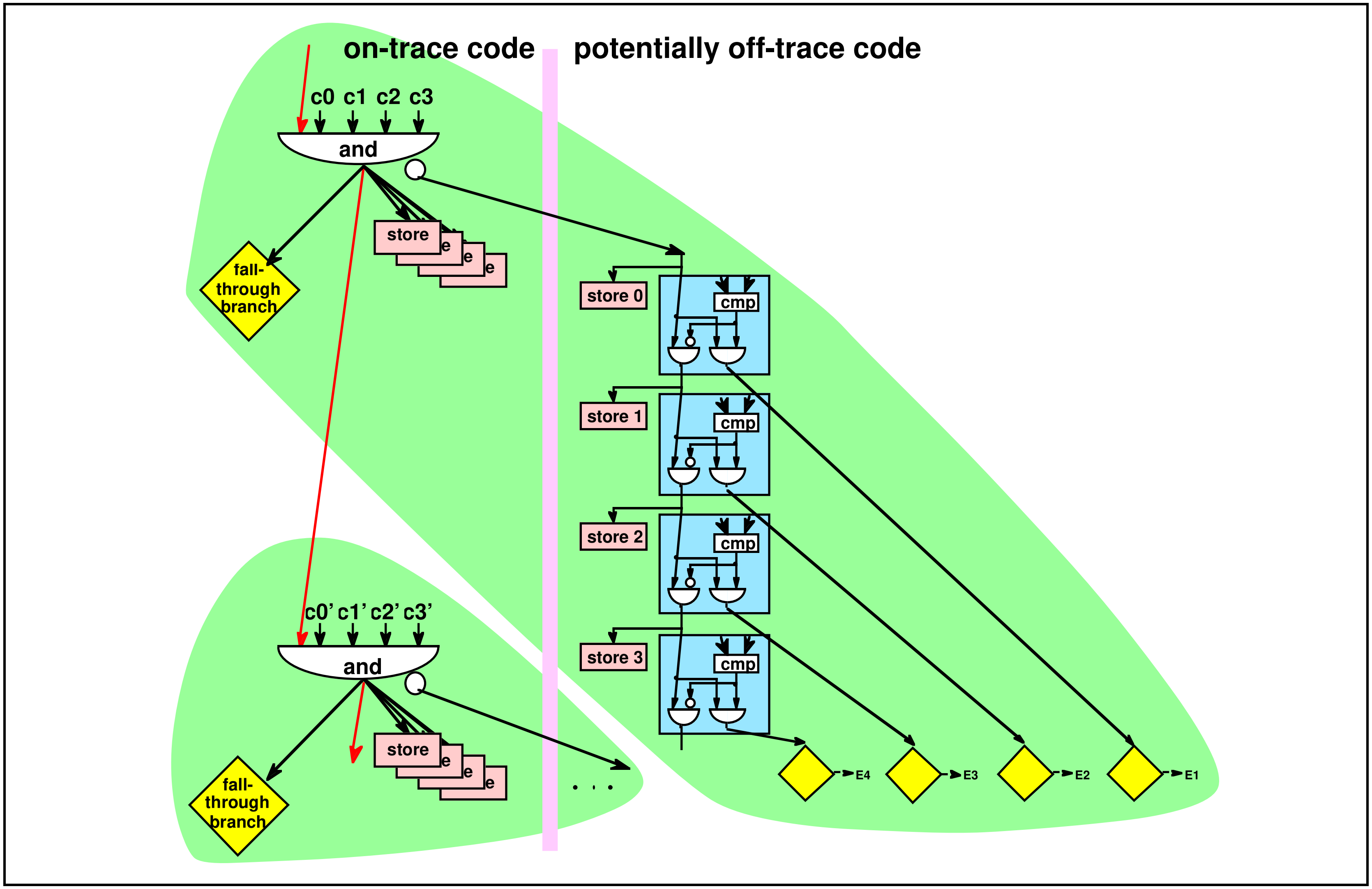
Data CPR



conventional

blocked back-substituted

Blocked Back-Substitution of FRP



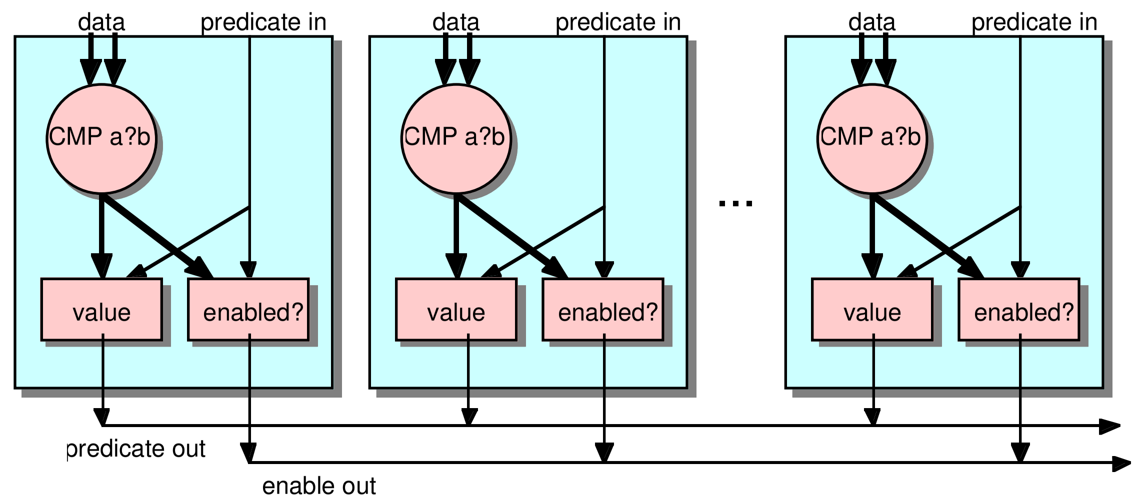
PlayDoh “Wired and” Expedites the Lookahead FRP

Result is initialized

Ops issued in any order

Ops may issue in same cycle

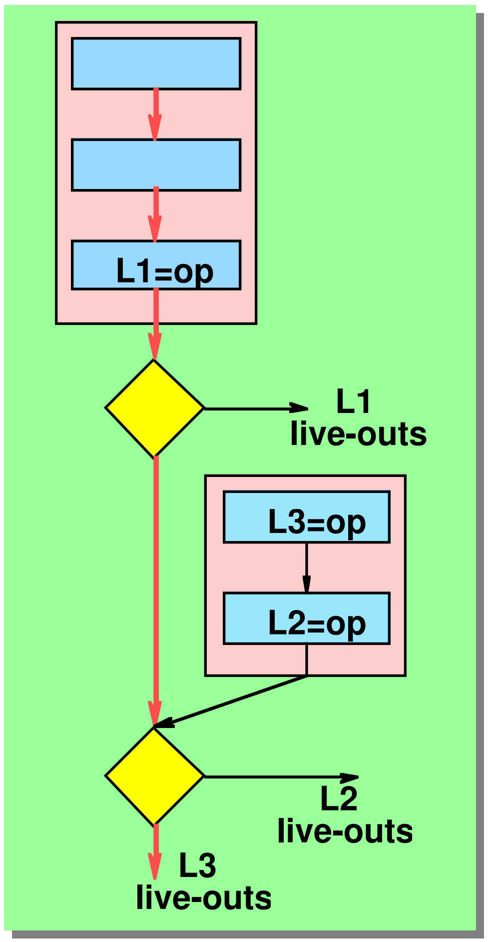
If they write the same value, the result is well-defined



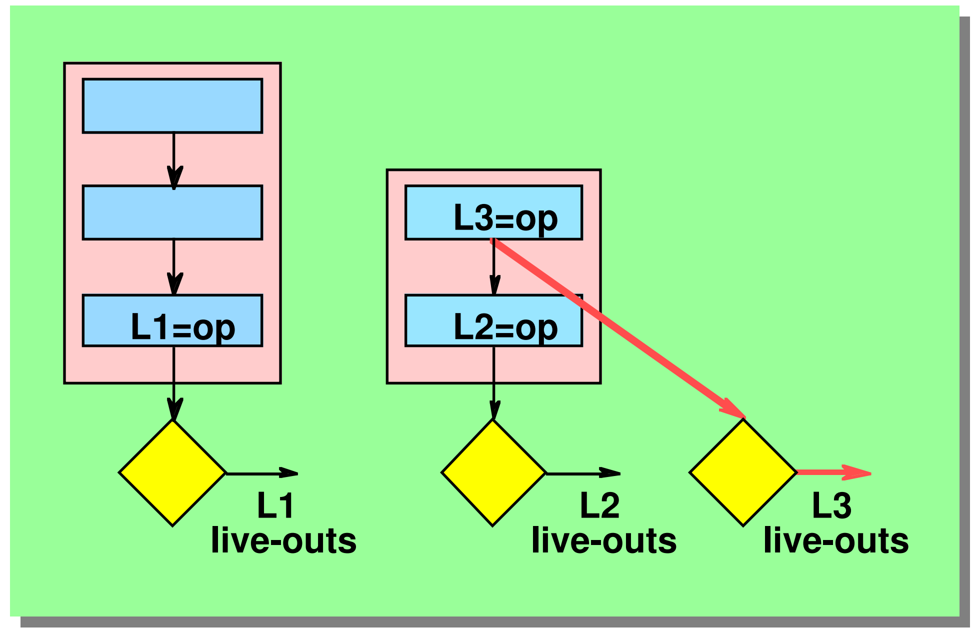
PlayDoh wired-and class compares - target action modifiers AN and AC

Predicate input	Result of comparison	AN (true)	AC (complement)
0	0	--	--
0	1	--	--
1	0	0	--
1	1	--	0

FRPs Alleviate Live-out Dependences

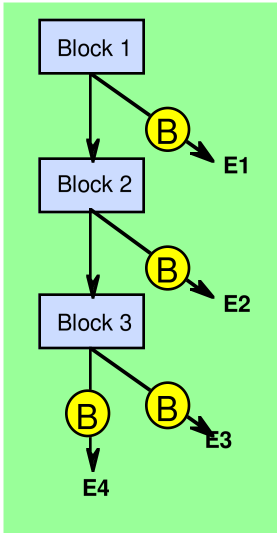


Superblock live-outs with ordered branches

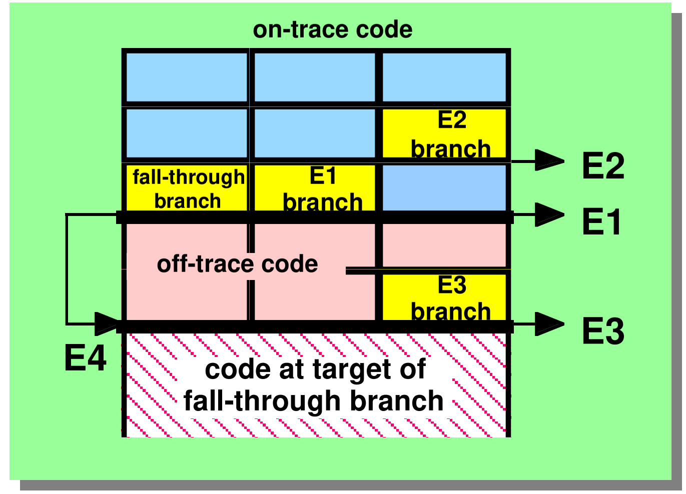


Superblock live-outs with unordered branches

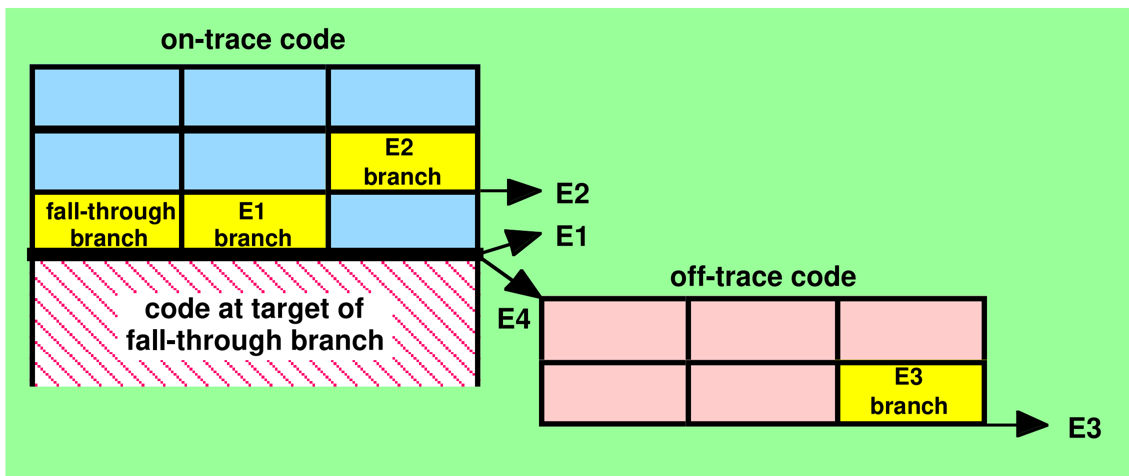
Scheduling Model for Use With CPR



Original code



Code after superblock scheduling



Reversing the fall-through branch

Benefits of Control CPR

Software transformations accommodate branch latency

Transformed code is efficient in operation count
- may result in fewer on-trace operations than original code
- fewer branches are traversed than in original code

CPR can be performed with or without predicate hardware

CPR can be hindered by:
- Memory aliasing
- Unpredictable branching

CPR Summary

Applicable to superscalar & VLIW processors

Applicable to loops (previous work)

Applicable to scalar codes (presented here)

The Promise:

Combined control and data CPR can move us from limited & fixed application ILP to scaleable ILP which saturates hardware