Timely, Efficient, and Accurate Branch Precomputation

Aniket Deshmukh, Chester Cai, Yale Patt
HPS Research Group

Background

- **Branch mispredictions still limit single-thread performance**
  - Most of these mispredictions come from a small set of problematic branches referred to as "hard-to-predict" (H2P) branches
  - Building extremely large predictors or using state-of-the-art neural networks at best reduces a third of the mispredictions
- **Alternative: Branch Precomputation**
  - Been around for over 25 years
  - Identify H2P branches and instructions in their dependence chains
  - Use the chains to compute H2P branch directions faster than the main thread
  - If the precomputation result arrives **by the time the corresponding branch is fetched**, it is used to override the branch predictor

Branch Precomputation: Prior Work

Key considerations:

accuracy **coverage** **timeliness**

Compiler techniques

- Create a perfectly accurate but heavy-weight helper thread
- Good coverage (>70%)
- Poor timeliness: <20% of precomputation results arrive in-time to override the prediction

Runtime solutions

- Create light-weight dependence chains for specific types of control flows
- Good timeliness: ~70% of precomputation results are timely
- Poor coverage (~30%)

Thus, the tradeoff between coverage and timeliness severely limits performance

A Timely, Efficient, and Accurate Precomputation Thread

We use precomputation results that arrive after the branch is fetched but before it is executed to issue early pipeline flushes

- Enabled by synchronized timestamps provided the thread construction mechanism

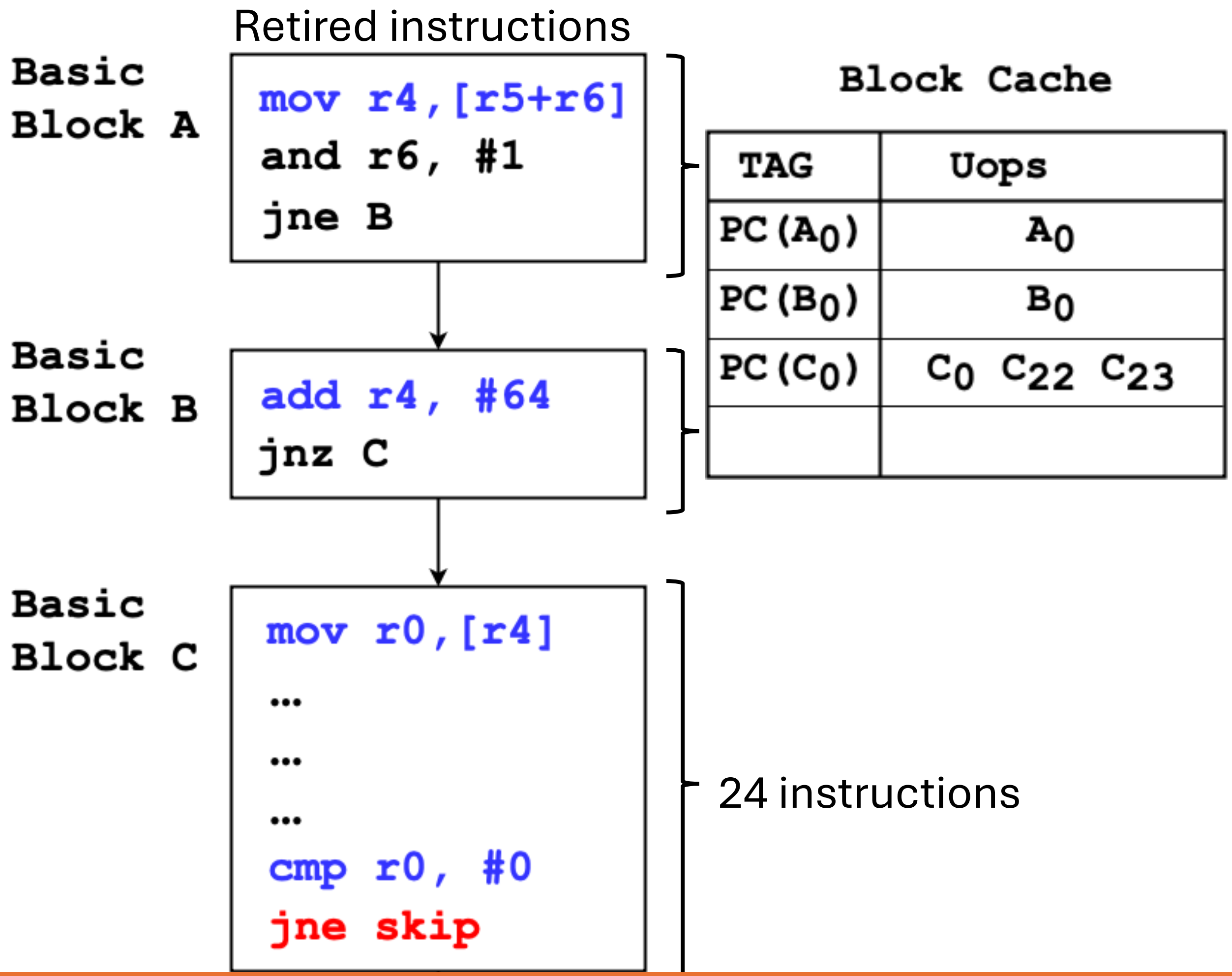
Mechanism for generating highly accurate dependence chains (>99.3%) at runtime for H2P branches

- Improves misprediction coverage without hurting timeliness, traces longer chains

Our precomputation thread can efficiently execute on-core without delaying the main thread significantly

The TEA thread provides a 10.1% performance improvement over a set of SPEC CPU2017 and GAP benchmarks

Identifying H2P Branch Chains

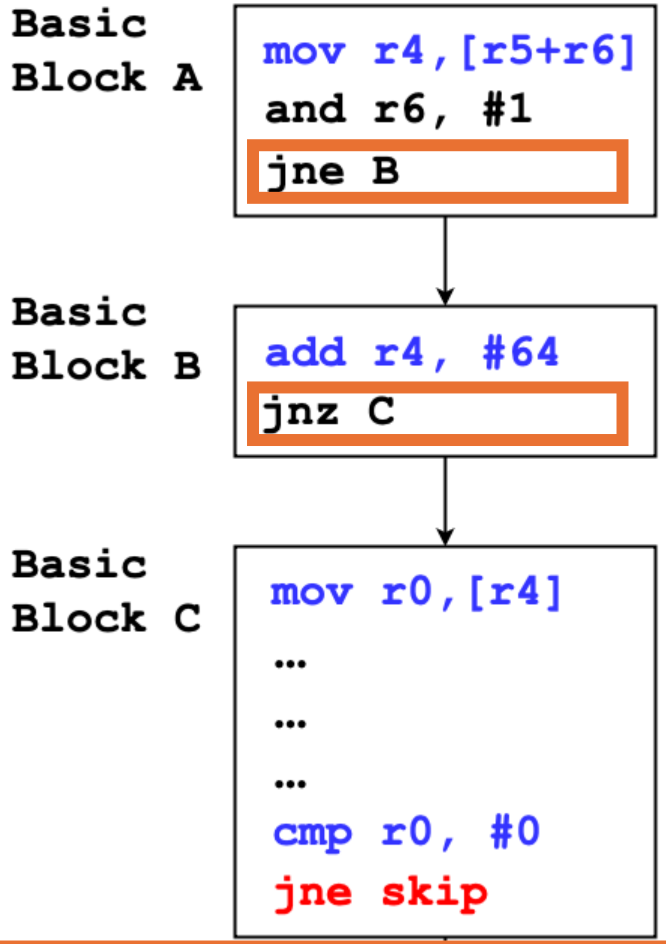


TAG	Uops
PC(A_0)	A_0
PC(B_0)	B_0
PC(C_0)	C_0 C_{22} C_{23}

- After retirement instructions are collected into a Fill Buffer
- Identify frequently mispredicting branches via the H2P Table
- Dependence chain instructions are traced via a Backward Dataflow Walk starting at these branches

Key idea: use the control flow sequence generated by the main branch predictor to stitch together block cache entries and re-construct the dependence chain at fetch time

Constructing the TEA thread



Block Cache

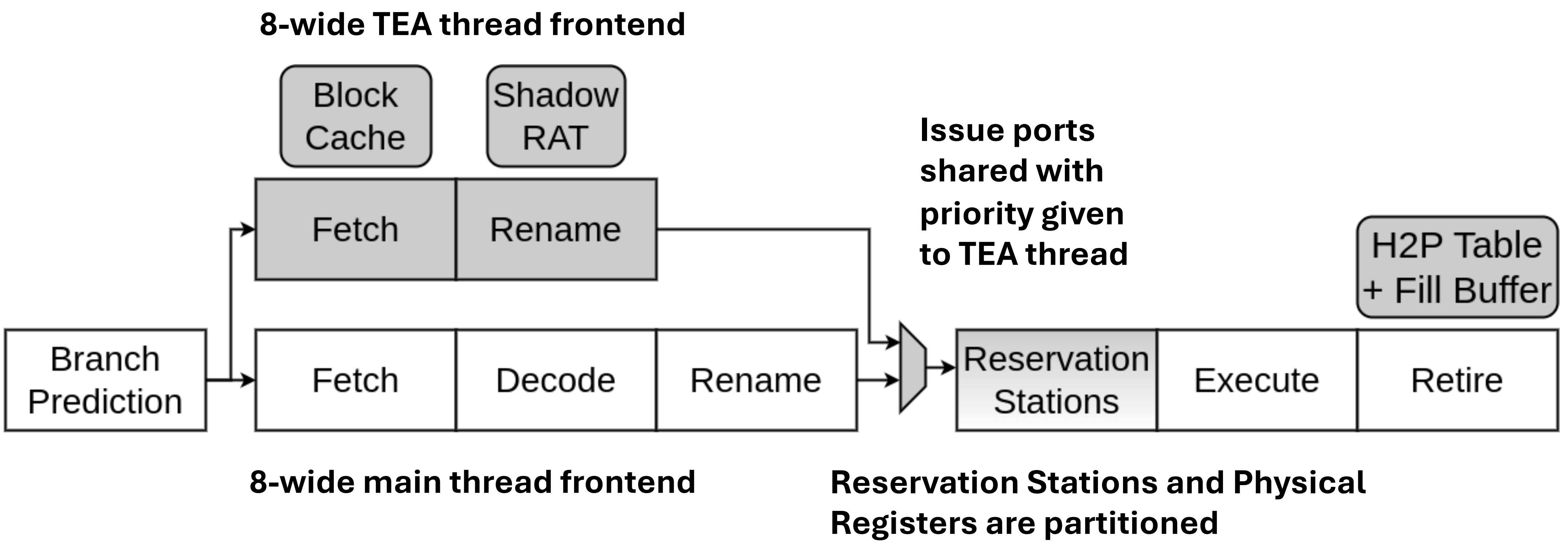
TAG	Uops
PC(A_0)	A_0
PC(B_0)	B_0
PC(C_0)	C_0 C_{22} C_{23}

Branch predictor generates the fetch address sequence:
A (3), B (2), C (24)

Main Thread	TEA Thread
A_0 A_1 A_2	A_0
B_0 B_1	B_0
C_0 … …	C_0 C_{22} C_{23}
… … …	
… C_{22} C_{23}	

- Both threads inherit the same branch IDs from the branch predictor
- Intermediate branches that are not hard-to-predict need not be precomputed

Implementation Overview



1. Faster fetch

2. Prioritized scheduling

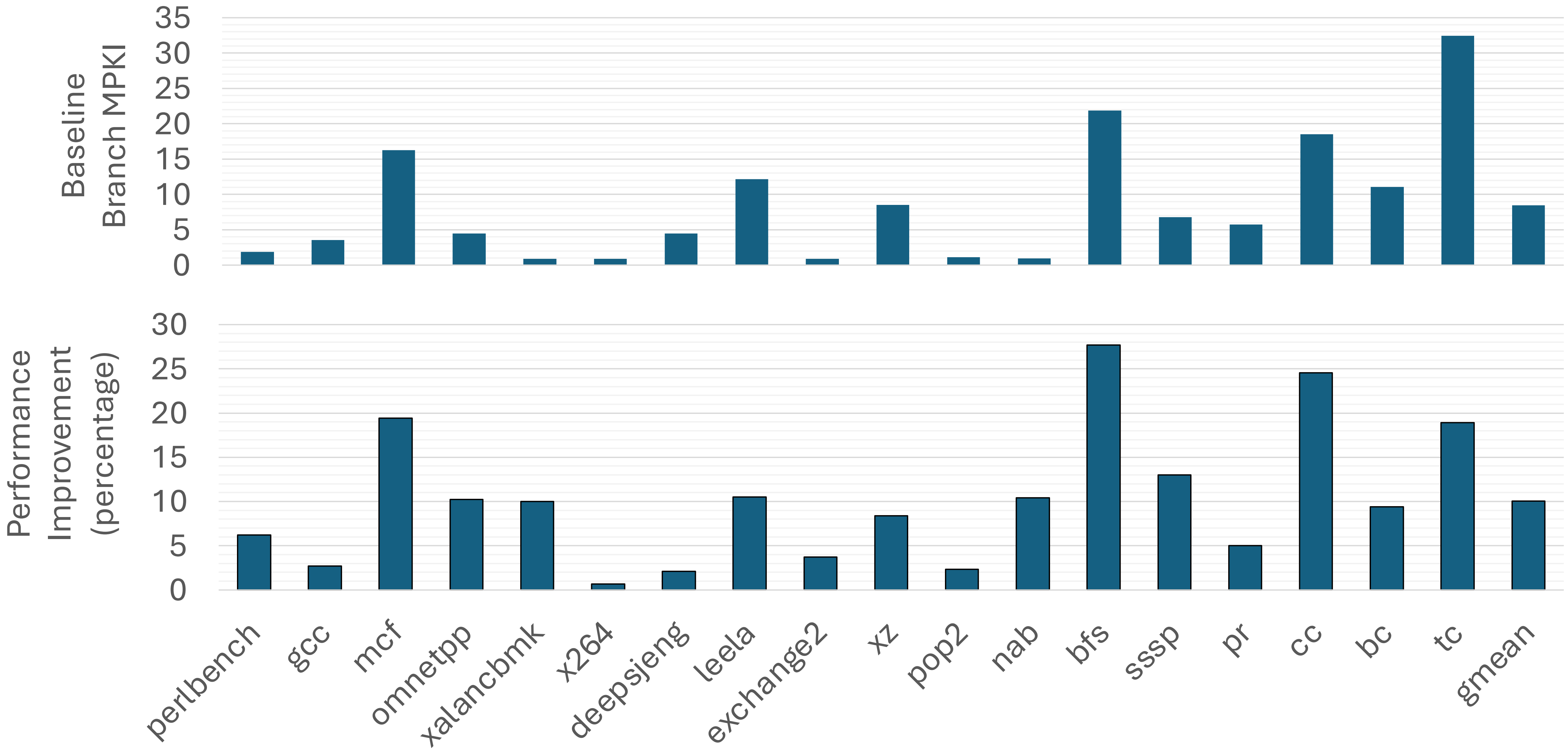
3. No backend stalls due to non-dependence chain instructions

Longer dependence chains improve timeliness as it allows the TEA thread to begin earlier

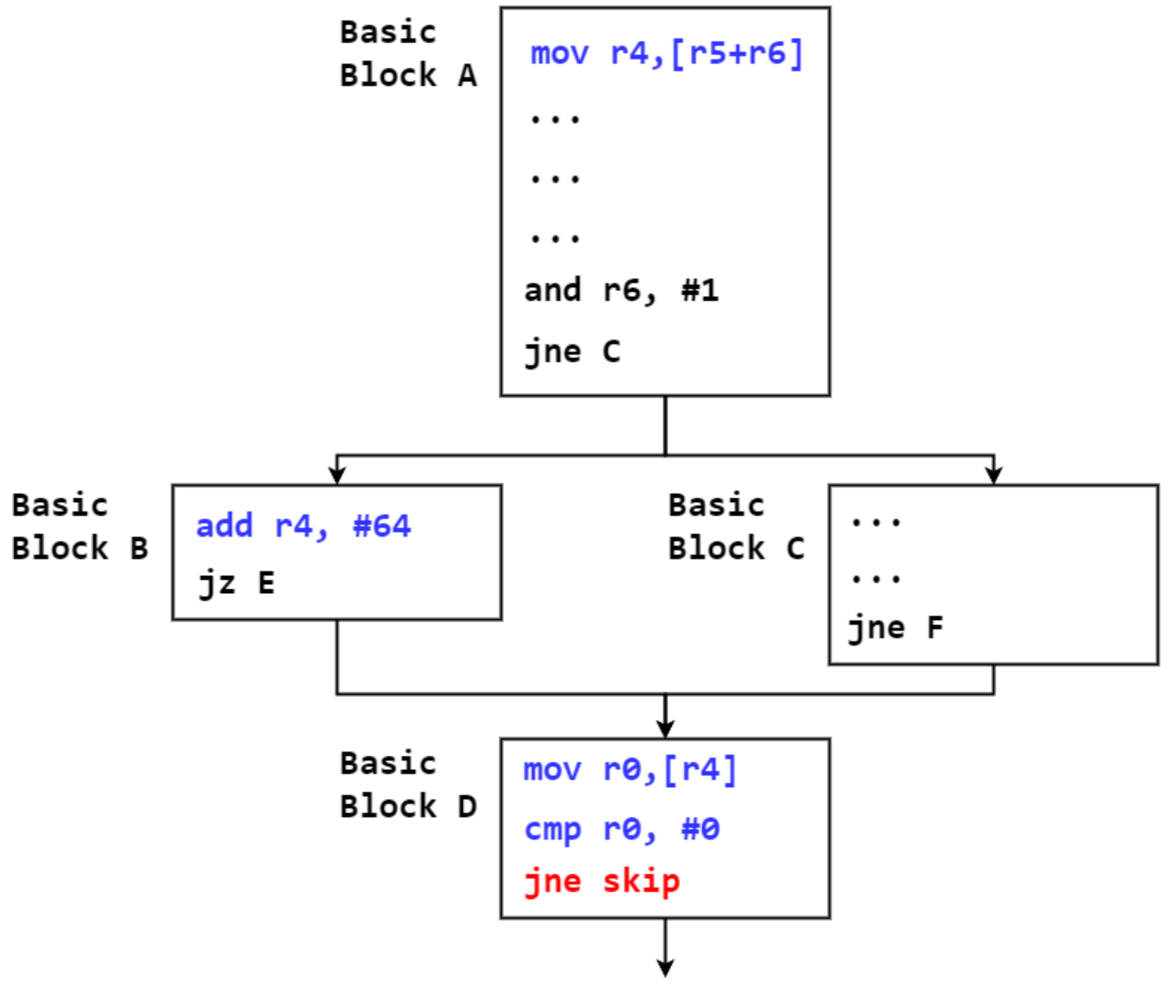
Thank you

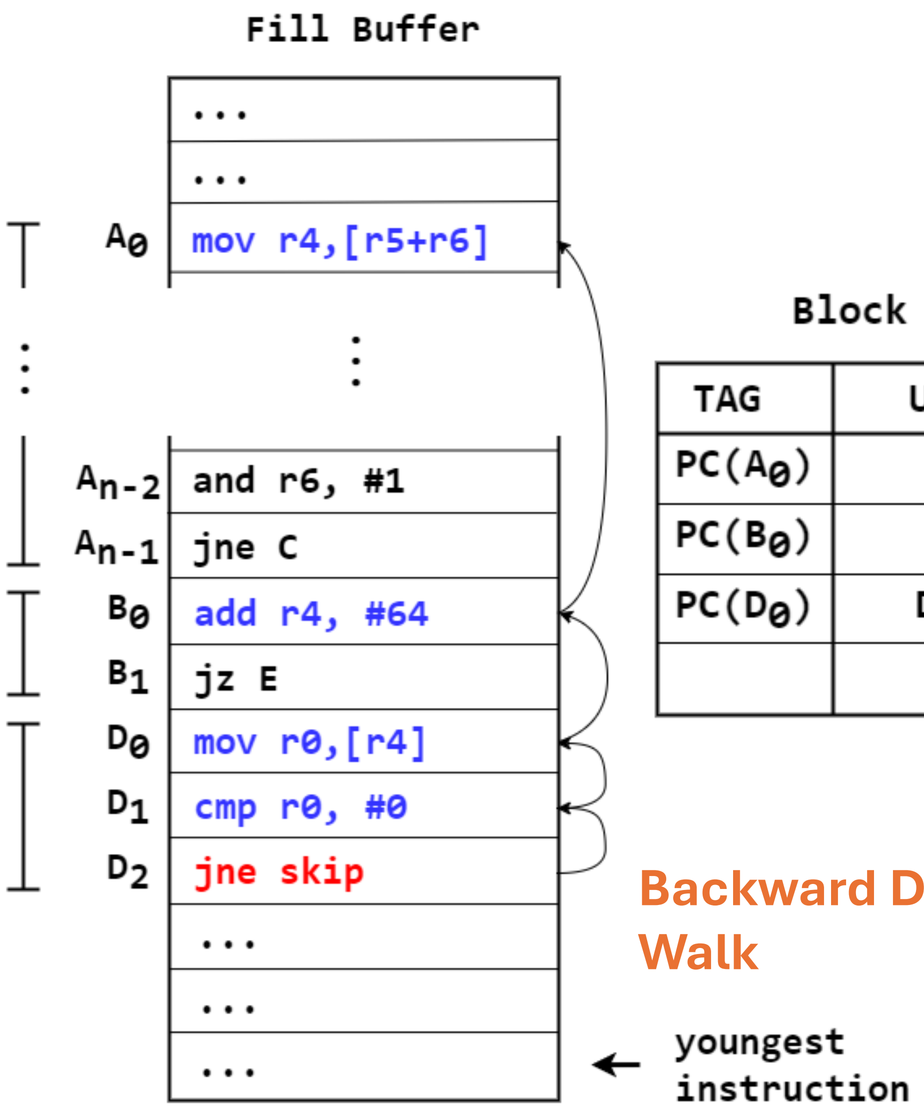
Aniket Deshmukh, Chester Cai, Yale Patt
HPS Research Group

Performance Improvement



Identifying H2P Branch Chains





Block Cache

TAG	Uops
PC(A_0)	A_0
PC(B_0)	B_0
PC(D_0)	D_0 D_1 D_2

Backward Dataflow Walk