Predicting the Usefulness of a Block Result: a μ-architectural Technique for High-performance Low-power Processors

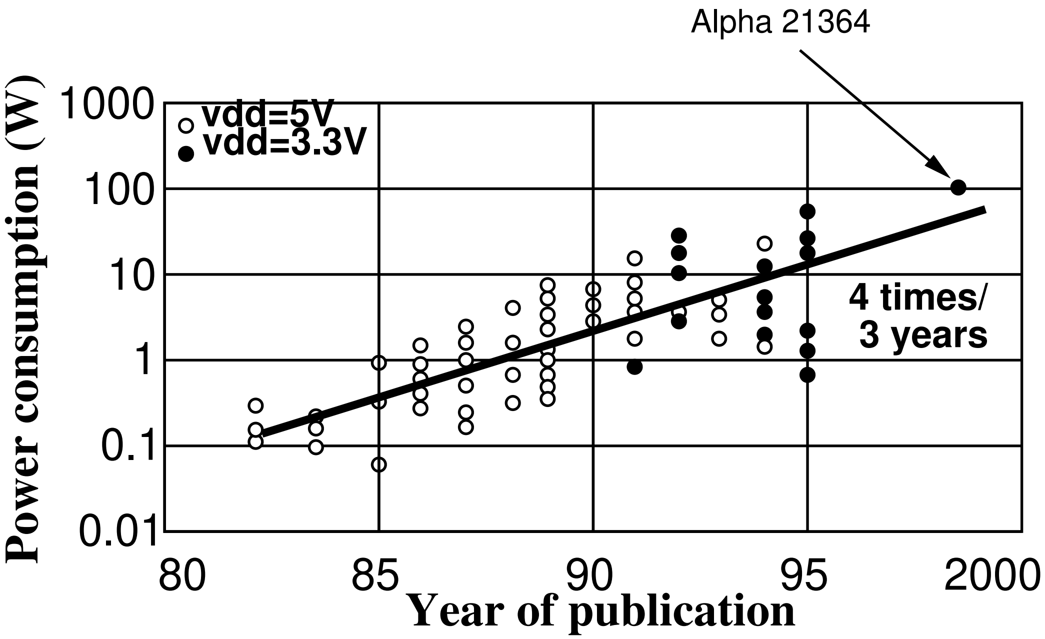
Enric Musoll

XStream Logic, Inc.
San Jose, CA

Outline

- Motivation
- Previous Work
- Predictors for Low Power: the Theory
- A Case Example
  - predictor for the L2
  - predictor for the BTB
- Conclusions

Motivation - Power Is Increasing



- Reliability
  - metal migration
  - electro-magnetic radiation
  - on-chip peak currents
- Package cost
  - ceramic package with air cooling
  - ceramic module with liquid cooling
- Battery-life

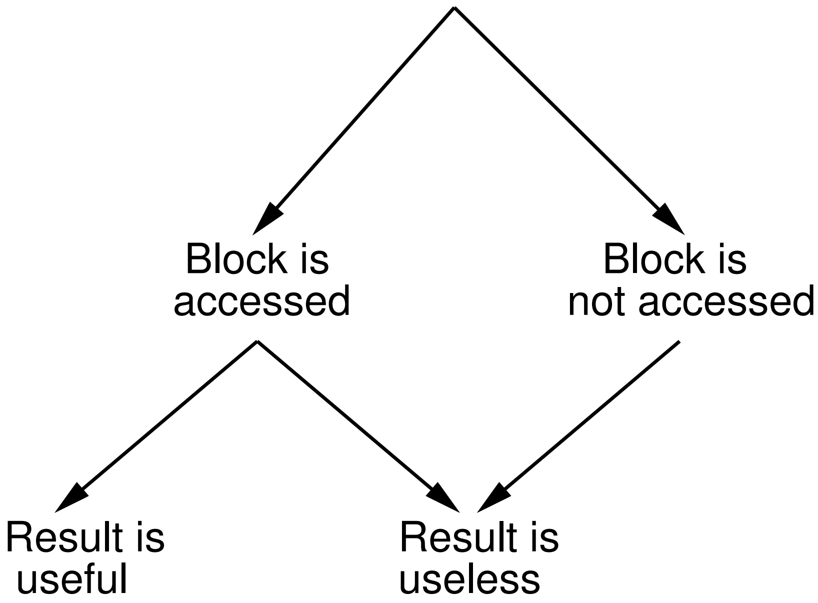
Motivation - Why Power Is Increasing?

- Higher clock frequencies
  - $W \propto f$
- Micro-architecture gets complicated
  - out-of-order engines
  - speculative techniques
  - on-chip level-two caches

High-performance micro-architecture techniques increase the power consumed by both committed and not-committed instructions

Motivation - Disabling Blocks

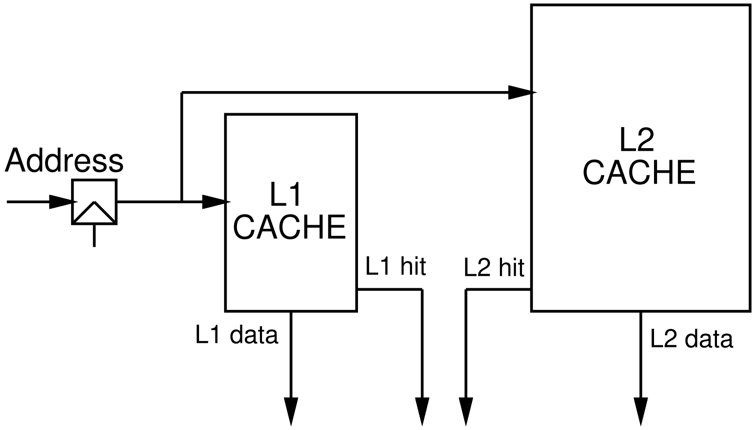
- Disable blocks when not used

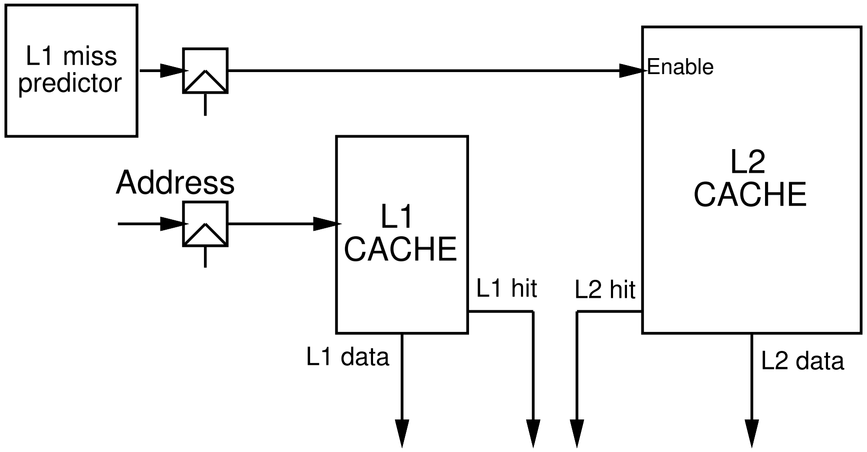


- Easy for some blocks (FUs)
- For others, not known till later

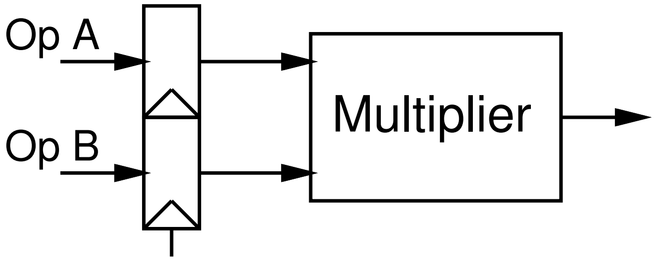
Motivation - To Know When To Disable Beforehand

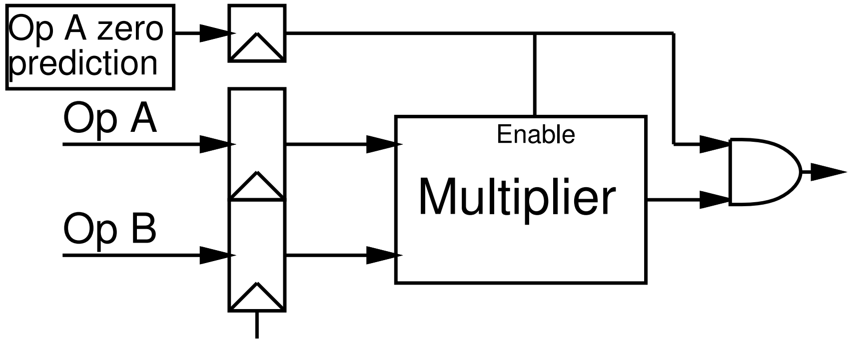
- On-chip L2 cache access





- Trivial operations in functional units





Motivation - Observations

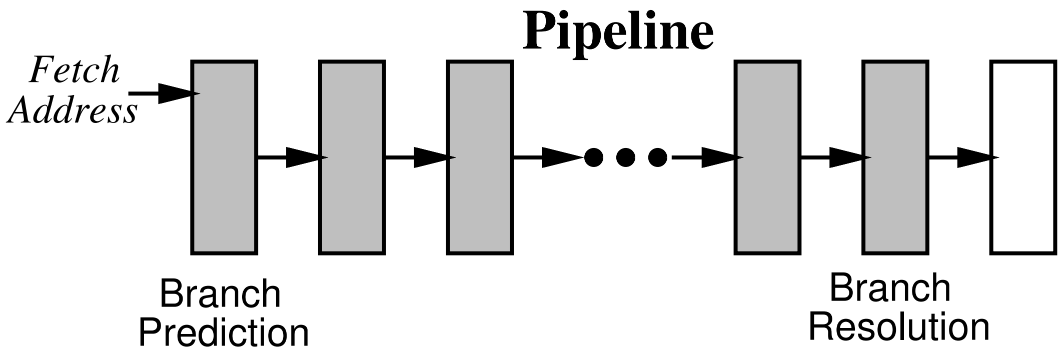
- Prediction for low power is based on

(1) a block is accessed for the execution of the current instruction but its result is often useless

(2) whether its result is useless or not known when the block starts to be accessed

Previous Work - Targetting Non-committed Instructions

- Non-committed instructions ⇒ useless work ⇒ disable their block accesses



- Confidence prediction of the branch predictor

Previous Work - Targetting Non-committed Instructions

- Patent # 5506976 [Jaggar'96]
- Patent # 5740417 [Kennedy & Croxton'98]

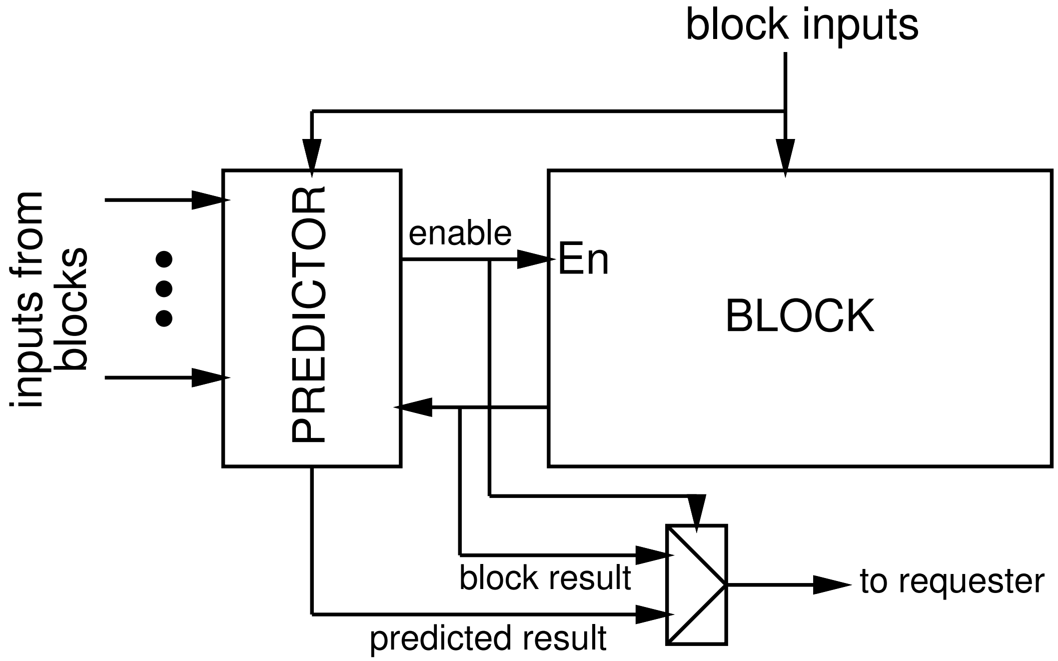
  Both disable the Branch Predictor Block only

- [Manne, Klauser & Grunwald'98]
  - Disable *pipeline stages*
  - 38% reduction in wrong-path instructions

- All focus on wrong-path instructions
- This work: also on committed instructions

Predictor For Low Power - Why Is It Needed?

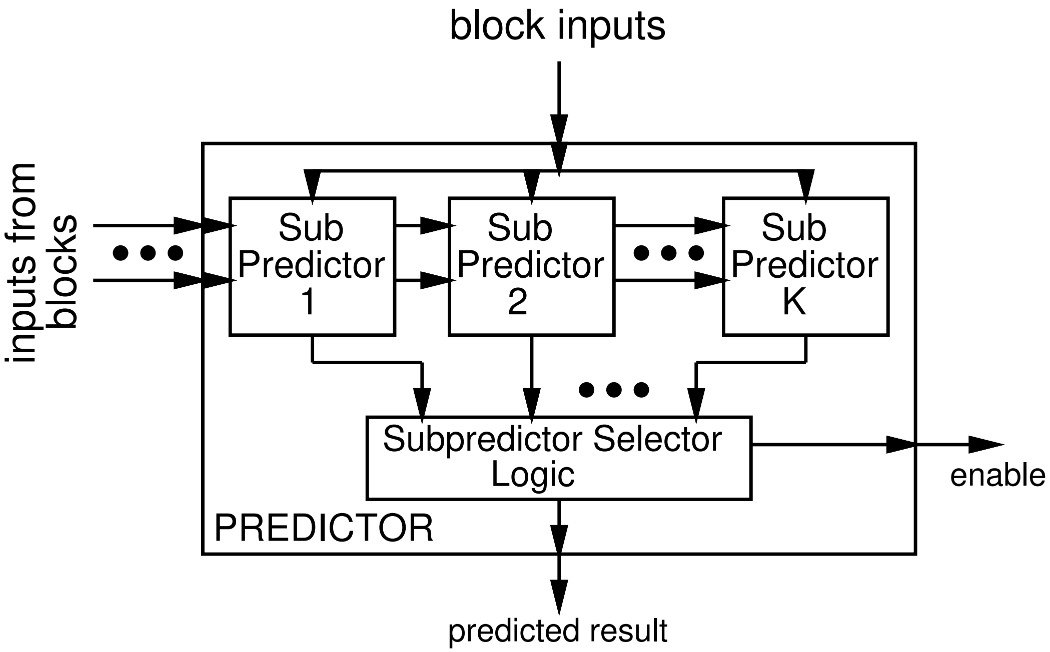
- Result of a block is not known beforehand to be useful



- The predictor may provide the result itself (predicted or actual)
  - Example: tiny L0 instruction cache

Predictor For Low Power - Sub-predictors

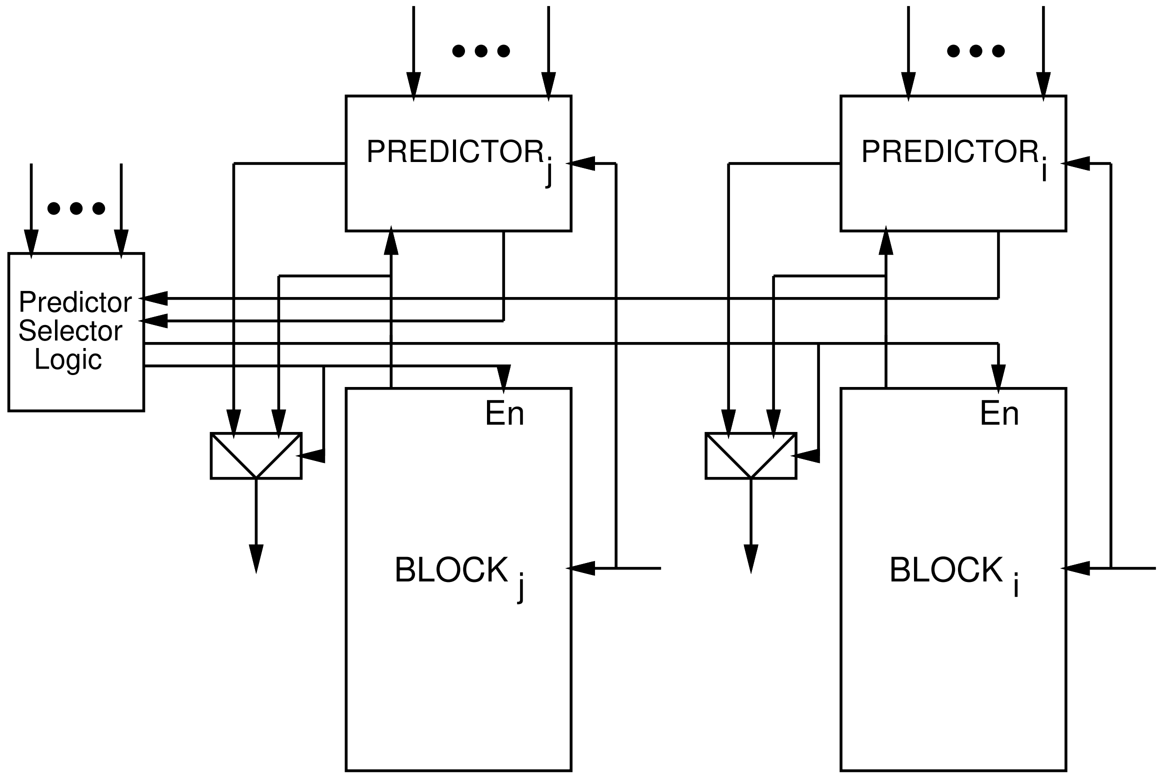
- Predicting the result of a block by independent predictors



- Sub-predictor Selection Logic
  - dynamic (≈ combined branch predictor)
  - static (simpler, less accurate, less power)

Predictor For Low Power - Multi-predictors

- Different predictors are associated to different blocks



- Predictor Selection Logic

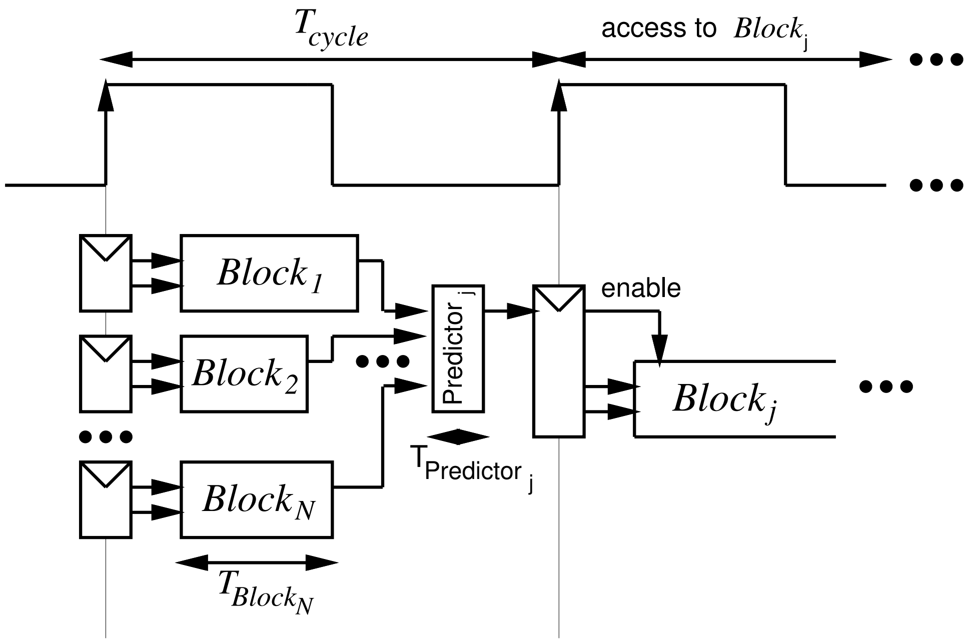
Predictor For Low Power - Multi-predictors

- Predictor Selection Logic: Strategies
  - *disable at least* X *blocks* (if $X > 0$, lower peak power consumption)
  - *disable at most* Y *blocks*
  - *disable any block*

- Or dynamically changing strategies

Predictor For Low Power - Drawbacks

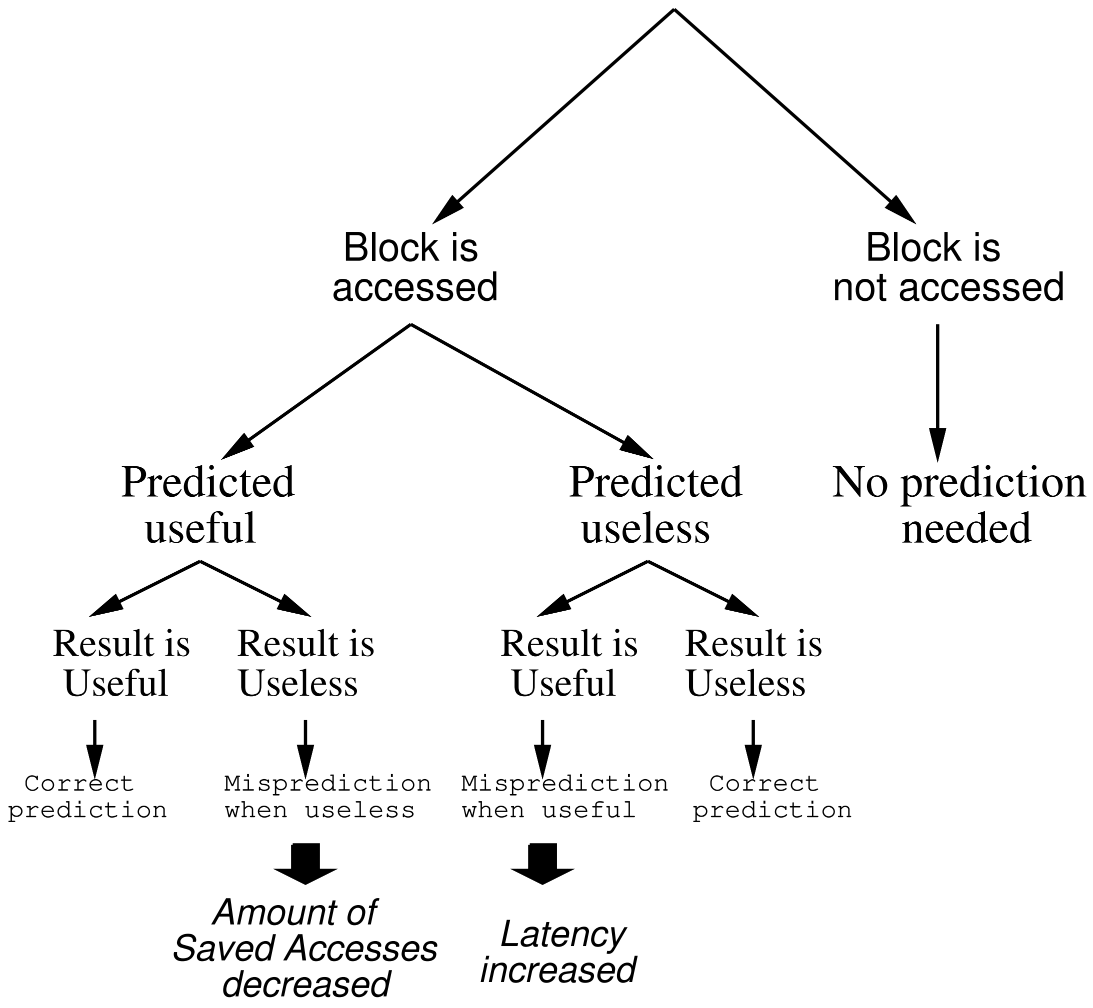
- Drawbacks
  - area
  - design & verification time
  - cycle time may increase



increases if $\quad Max(T_{Block_i}\ i = 1..N) + T_{Predictor_j} > T_{cycle}$

Predictor for Low Power - Mispredictions

- Consequences of mispredictions



Predictor for Low Power - Overall Power Reduction

- Overall micro-processor power consumption reduction

  (1) *how accurate the predictors are*

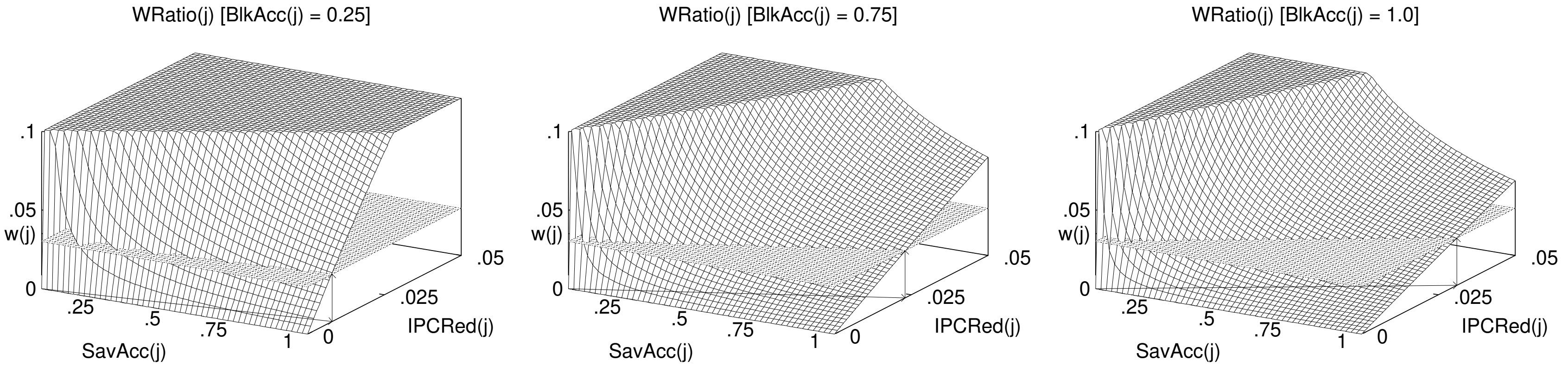
  (2) *percentage of the total processor power consumption that correspond to the blocks being predicted*

  (3) *how sensitive to the IPC the different blocks are*

Predictor for Low Power - Design Space

- Accuracy: $\texttt{SavAcc}^j$
- Effectiveness: $\texttt{WRatio}^j$
- Sensitiveness: $\Delta\texttt{IPCRed}^j$, $\texttt{BlkAcc}^j$

$$\texttt{WRatio} = f(\texttt{SavAcc}, \Delta\texttt{IPCRed}, \texttt{BlkAcc})$$



Case Example - High-Performance Processor

- High-level out-of-order micro-processor
  - on-chip L2 cache (instruction fetches only)
  - Branch Target Buffer

- Focus on the number of block accesses saved
  - significant amount of saved accesses $\Rightarrow$ significant power reduction

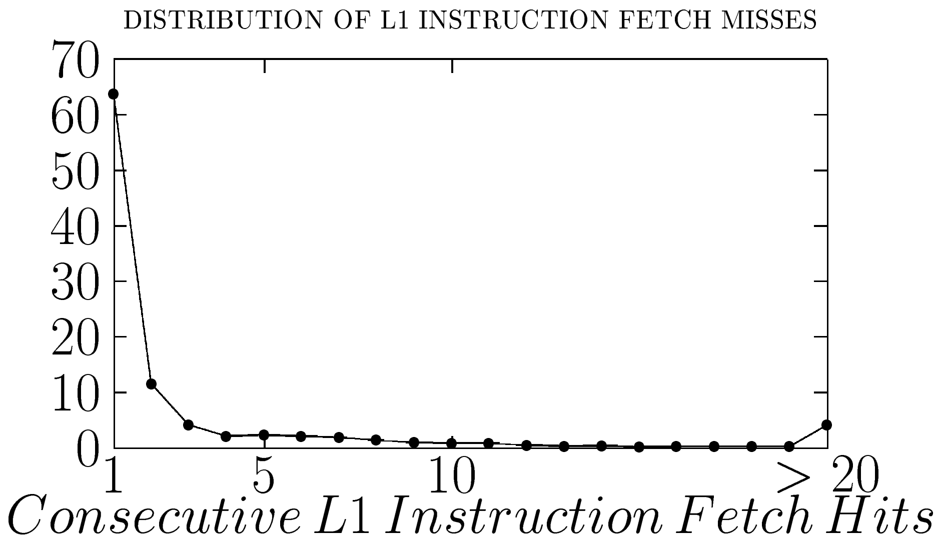
Case Example - On-chip L2 Cache

- Predictor behavior:

  (a) no L1 miss within next W accesses $\Rightarrow$ predict L1 miss until next miss

  (b) otherwise $\Rightarrow$ predict L1 hit

  (predict L1 hit $\Rightarrow$ disable L2)



(85% of all misses occur within a distance of 5 or less)

Case Example - Branch Target Buffer

- (Small) Pentium Pro BTB accounts $\approx$ 5% of overall processor power
- Two predictors $\Rightarrow$ Sub-predictor Selection Logic
  - First: based on BTB miss bursts
  - Second: based on predictability of loops

Case Example - Loop Predictor for BTB

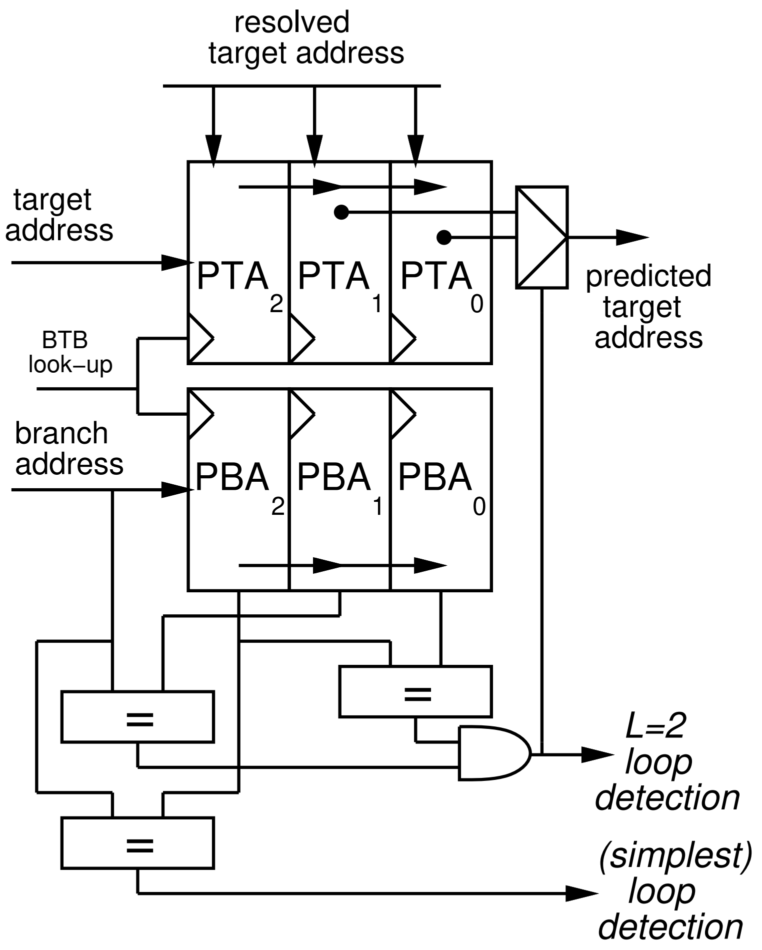
ini_loop:
- ...
- **if** $cond_1$ **goto** $target_1$
- ...

$target_1$:
- ...
- **if** $cond_2$ **goto** $target_2$
- ...

$target_2$:
- ...
- **if** $cond_{L-1}$ **goto** $target_{L-1}$
- ...

$target_{L-1}$:
- ...
- **if not** $cond_{end\_loop}$ **goto** ini_loop

end_loop:
- ...



- Provides the predicted target address if detects a loop

Case Example - Sub-predictor Selection Logic

- Two BTB Predictors ⇒ Sub-predictor Selection Logic

`BTBMiss` `Predictor` predicts	`Loop` `Predictor` predicts	**SSL Action**
hit	*no loop*	Enables BTB
miss	*no loop*	Disables BTB. Predicted target address is the fall-through path
- -	*loop*	Disabled BTB. Predicted target address is provided by `LoopPredictor`

Case Example - Simulation Scenario

- SimpleScalar, SPEC95
- 10M + 100M instructions/program
- Weighted average
- Micro-architectural parameters

  8KB DM 1-cycle L1 instruction and data caches
  256KB 4-way 6-cycle unified L2
  64-entry 4-way instruction TLB
  128-entry 4-way instruction TLB
  2K-entry 4-way BTB, Bimodal branch predictor
  4 simple, 1 complex integer FUs
  4 simple, 1 complex FP FUs
  2 L1 cache ports

- On-chip L2 Cache
  - `WindowBasedPredictor`: W
- Branch Target Address
  - `BTBMissPredictor`: M
  - `LoopPredictor`: L

Case Example - Single-predictor/Sub-predictors Results

Block	**Predictor**	**Parameters**	**% IPC Red.**	**% Saved Accesses**
L2	`Window`	$W = 8$	0.5	76.5
		$W = 16$	0.3	71.0
		$W = 32$	0.2	64.7
BTB	`BTBMiss`	$M = 1$	3.6	5.7
		$M = 2$	1.0	1.8
		$M = 3$	0.5	0.8
	`Loop`	$L = 1$	0.7	4.8
		$L = 2$	2.4	9.5
		$L = 3$	3.0	11.3
	Both	$M = 1, L = 1$	4.3	10.5
		$M = 2, L = 2$	3.5	11.3
		$M = 3, L = 3$	3.5	12.1

- `WindowBasedPredictor` degrades IPC the least
- `LoopPredictor` degrades IPC less than `BTBMissPredictor`
- When `LoopPredictor` and `BTBMissPredictor` are both active
  - %`SavAcc` $\leq$ %`SavAcc` of both ; %Δ`IPCRed` $\leq$ %Δ`IPCRed` of both

Case Example - Multi-Predictor Results

- Predictor Selection Logic results

PSL Strategy *disable ...*	**% IPC Degradation**	**% L2 Saved Accesses**	**% BTB Saved Accesses**
at least one	7.6	98.3	11.3
at most one	1.1	70.0	1.4
any	3.7	70.0	11.3

$(W{=}16,\ M{=}2,\ L{=}2)$

- With random predictors, 10.7% IPC reduction instead of 3.7%

Case Example - Multi-Predictor Results

- Power consumption in high-performance processors keeps increasing in spite of aggressive circuit design techniques and process shrinks.

- Micro-architectural technique to reduce the amount of useless accesses to blocks of the micro-processor performed by instructions of both the wrong and correct path.

- A case example shows that significant reduction in the number of accesses is possible with the presented methodology.