先進計算機構成論 付録：分岐予測の詳細

東京大学大学院 情報理工学系研究科 創造情報学専攻
塩谷 亮太
shioya@ci.i.u-tokyo.ac.jp

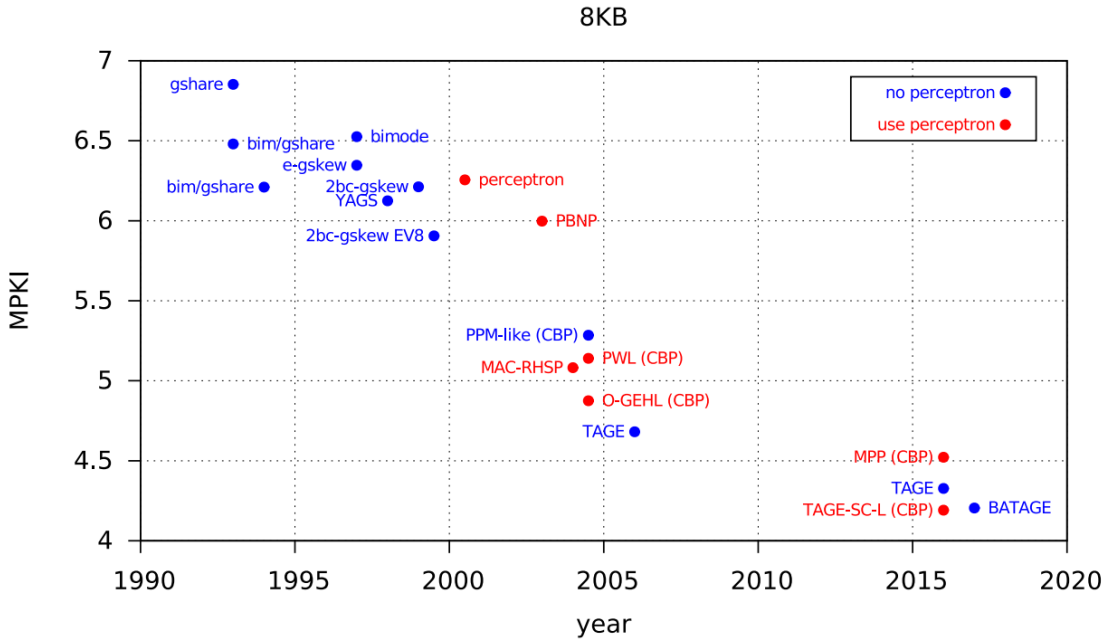
はじめに

- この資料では，講義の範囲を超えるような分岐予測の詳細な話題についてまとめます
  - ◇ 最新の分岐予測器
  - ◇ 複数命令同時フェッチ時の実装

現代の分岐予測器の性能（8KB 使用の場合）

PIERRE MICHAUD, An Alternative TAGE-like Conditional Branch Predictor, TACO 2018 より

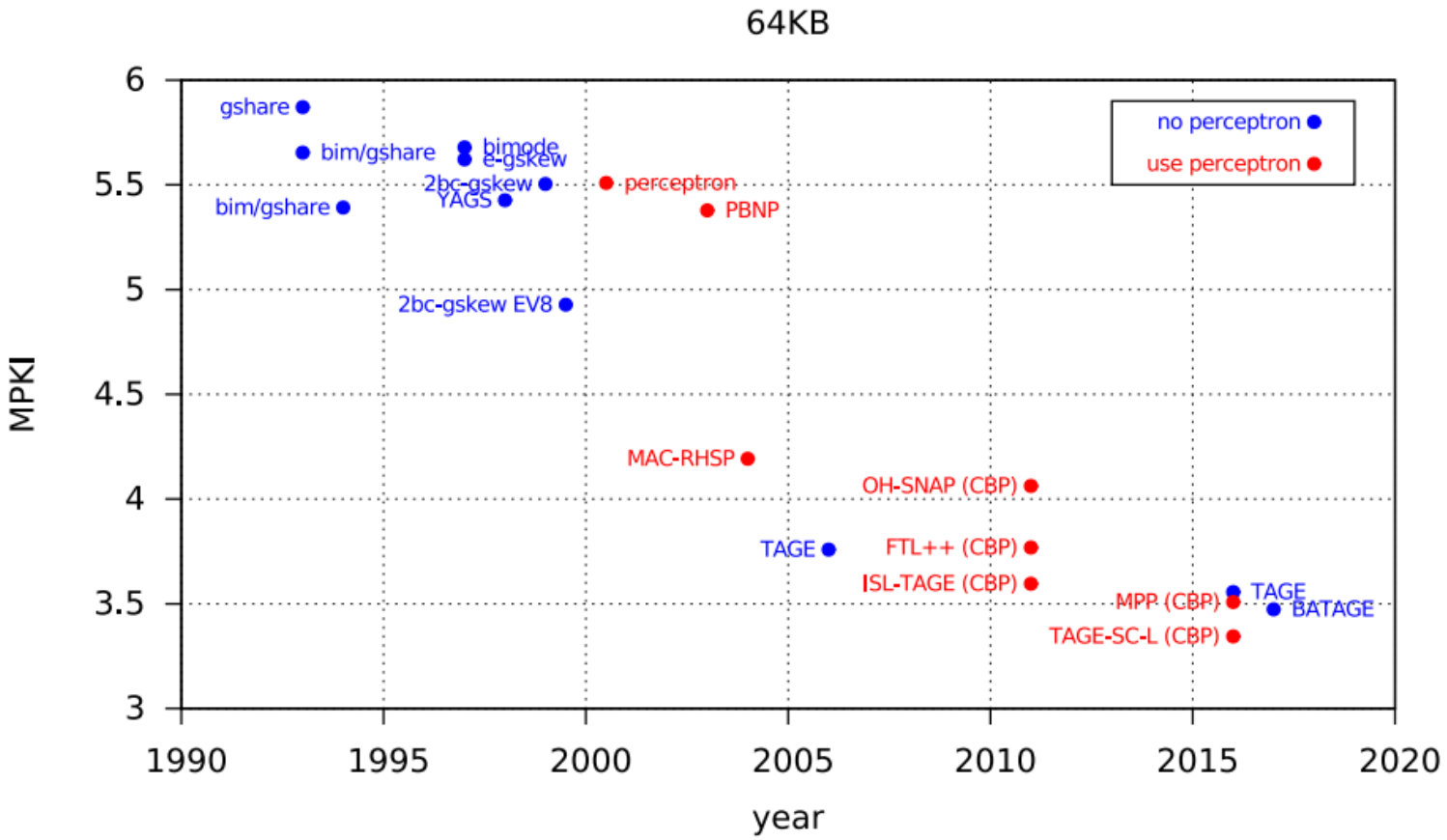
- Championship Branch Prediction (CBP) 2016 の環境を使って評価
  - https://www.jilp.org/cbp2016/
- 基本的には TAGE 系が一番強い
  - TAGE 自体もいろいろ改良されている
    - 統計的補正，テーブルのインタリーブなどが効果が大きい
  - 今日のポイント：右下の BATAGE が簡単かつ性能がよい



現代の分岐予測器の性能（64 KB 使用の場合）

PIERRE MICHAUD, An Alternative TAGE-like Conditional Branch Predictor, TACO 2018 より

- 8KB → 64KB で MPKI が 1 減るぐらい

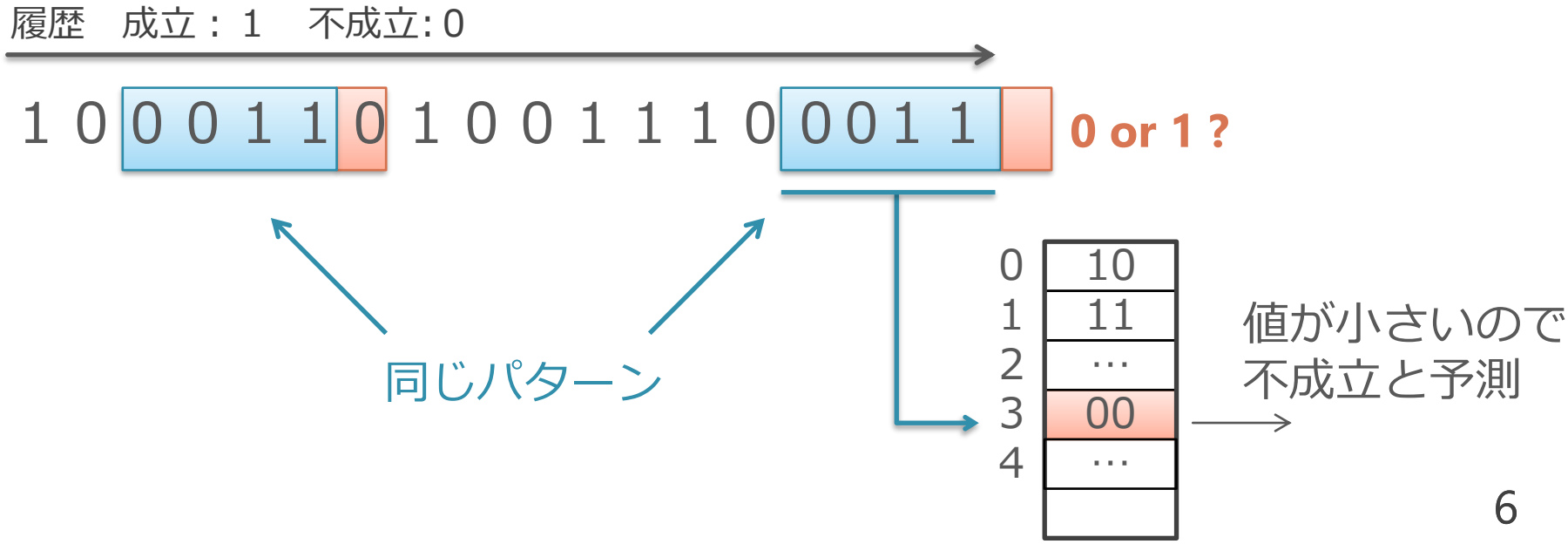


分岐予測

1. 方向分岐予測器
    1. 分岐履歴を使った予測器の基本と問題
    2. パーセプトロン予測器
    3. ハッシュ・パーセプトロン予測器
    4. TAGE 予測器
    5. BATAGE 予測器
2. TAGE の実装
3. 間接分岐予測器
4. 複数命令同時フェッチ時の予測の実装方法

履歴を用いた予測器の基本

- 基本的なアイデア：分岐方向の履歴をビット列で表す
  - ◇ 履歴のビット列をインデクスとしてテーブルにアクセス
    - □ テーブルのエントリは飽和型カウンタ
    - □ 成立時にインクリメント，不成立時にデクリメント
  - ◇ 直前の履歴でテーブルをひく
    - □ 直前に同じパターンがくると，同じエントリにアクセスする
    - □ 二進数で 0011 = 表の3番目のエントリ



履歴を用いた予測器

- 履歴の保持方法/作り方の違いで，いろいろな方法がある

1. ローカル履歴予測器
   - PC ごとに履歴を保持

2. グローバル履歴予測器
   - PC を区別せず履歴を保持
   - 基本的にこちらが使われる事が多い
     - ローカル履歴で予測可能なものを内包できる
   - 以降の話はこちらを前提

履歴長と予測精度

- 一般に，グローバル履歴長を長くするほど精度はあがる
  - ◇ より遠い分岐の相関が拾えるようになる
- 履歴長が1000以上のところに相関がある場合もある
  - ◇ 関数呼び出しやループをまたいだところにある分岐間の相関など

```
if (a > 0) {...}
b++;
c--;

// 1つ前の分岐と相関
//  （毎回方向が逆）
if (a <= 0) {...}
```

```
if (a > 0) {...}

// 関数内で分岐を
// 999 命令実行
function();

// 1000 個前と相関
if (a <= 0) {...}
```

```
if (a > 0) {...}

for (i...N) {
    if (...) {...}
}

// N 個前と相関
if (a <= 0) {...}
```

履歴長と予測精度

- 実際にはハードウェア（特にテーブルの大きさ）の制約がある
  - 短時間にアクセス可能な大きさに限られる
    - 最大数KBから数十KB程度？
  - 履歴長に対し，２の累乗のオーダーでエントリ数が増加
    - インデクスの最大値 = 2^履歴長 – 1
    - ハッシュ関数で折りたためばインデクス最大値は一定内におさまるが，出現パターン数が増えることには違いがない

パーセプトロン予測器

- パーセプトロンを使う分岐予測器は２タイプある：
  1. 狭義のパーセプトロン予測器
     - グローバル履歴のビット列を単層パーセプトロンにかける
  2. ハッシュ・パーセプトロン 予測器
     - 異なる履歴長を使って複数のテーブルから重みを読み出す
     - 読み出した重み自体を加算
- 備考：
  - 塩谷が学生の頃は半分ネタだと思われていたが，今は実用化済み
  - （正直，塩谷はパーセプトロン一般の事をそんなにわかっていない

狭義のパーセプトロン予測器

1. 最初に提案されたパーセプトロン予測器
   （以降の数ページでは，これを説明）
   Jimenez, Daniel A et al., Dynamic branch prediction with perceptrons, HPCA 2001

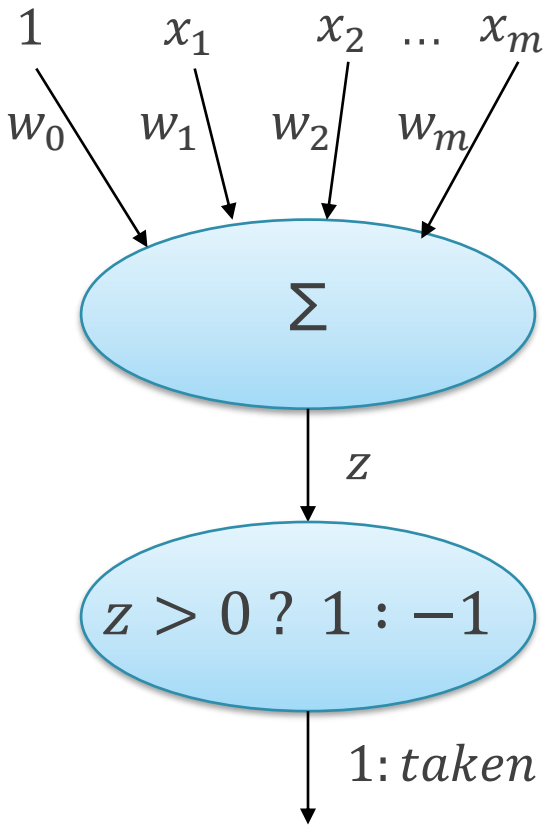
2. Path-Based 予測器
   Jimenez, Daniel A et al., Fast Path-Based Neural Branch Prediction, MICRO 2003

3. Piecewise Linear 予測器
   Jimenez, Daniel A et al., Piecewise Linear Branch Prediction, ISCA 2005

パーセプトロン予測器

- モチベーション：
  - グローバル履歴のうち，本当に相関があるのは一部のビットのみ
  - ある特定の if 文同士で相関がある場合，間の履歴は無駄
- パーセプトロン予測器：
  - 単層パーセプトロンを使って予測
  - 高速に予測を行う必要があるため，ややこしいことは無理
    - １層限定で，重みは８ビット固定小数点とか

単層パーセプトロン
（通常は 0,1 が入力だが，分岐予測では-1,1に



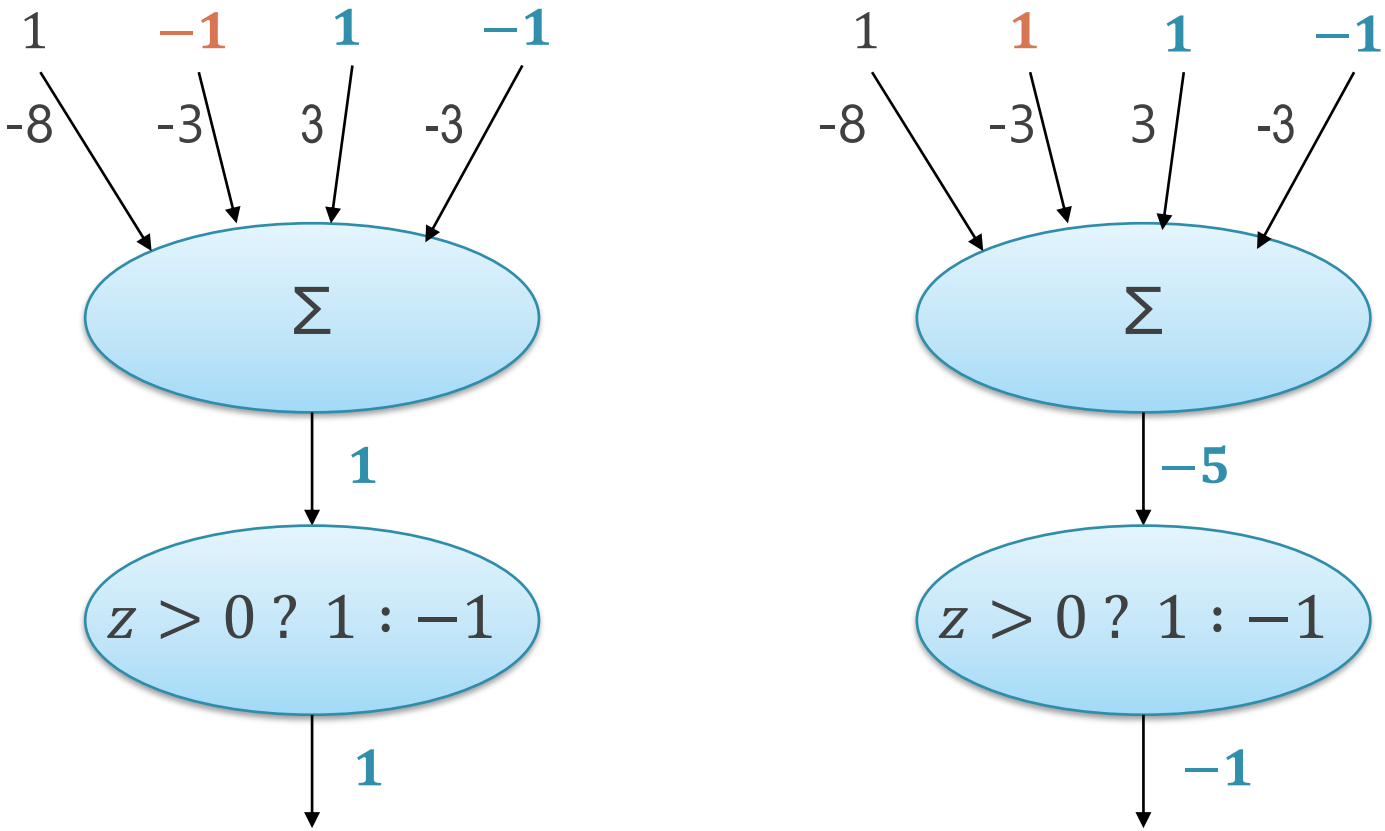
- $z = w_0 + \sum_{i=1}^{m} w_i x_i$
  （w_0 はバイアス）

◇ 入力：x_1 から x_m
（1:taken or -1:untaken）

◇ 重み（アナログ値）：w_0 から w_m（w_0 はバイアス）

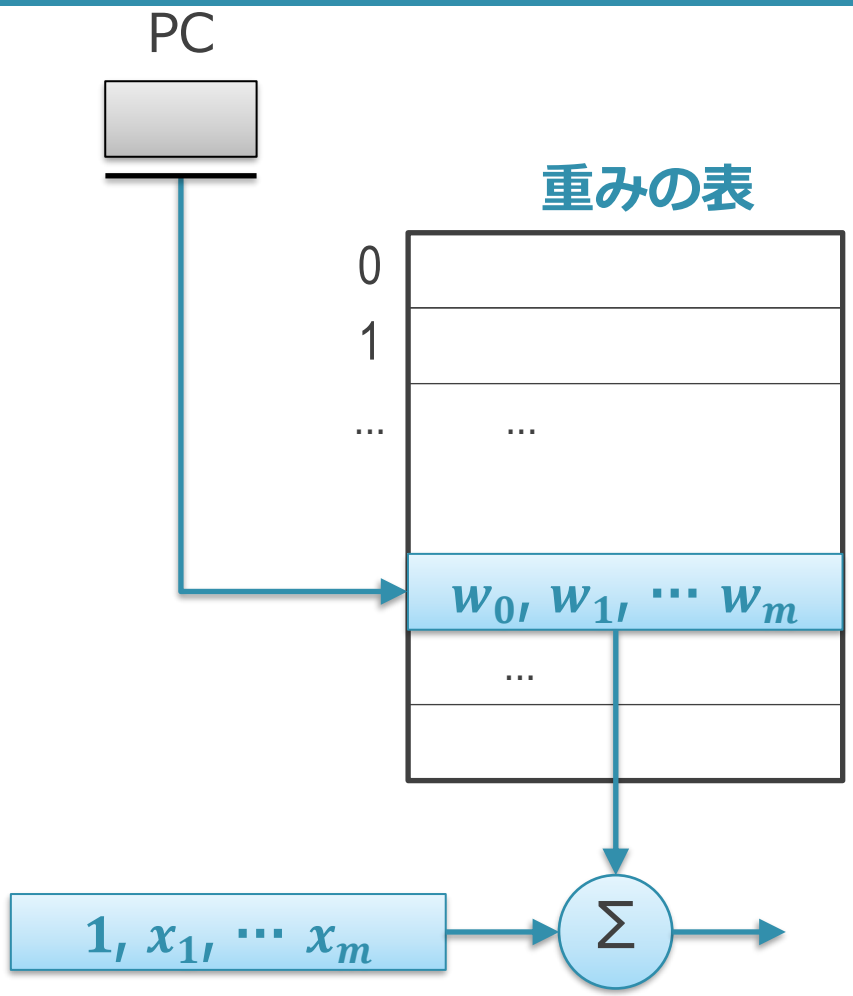
◇ 学習：$w_i = w_i + t \times x_i$
（実際の分岐方向 t が 1 の時，x_i が 1 だったなら w_i を大きく
-1 だったなら w_i を小さくする

「-1,1,-1 なら 1 」を学習させた場合



- -1,1,-1 以外のパターンが来た場合は，z は負の方向に
- 左端はバイアス
  - ◇ $z > 0$? は，$\sum_{i=1}^{m} w_i x_i > -w_0$? に等価

最初に提案されたパーセプトロン予測器



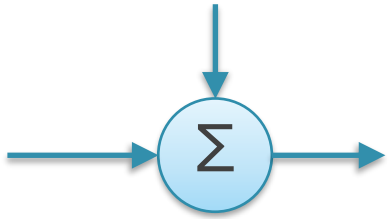
- PC の一部をインデクスとして重み表をひく
  - その PC に対応した重みのセットがとれる
- あとはグローバル履歴を x_i としてパーセプトロンの処理を行う

実際の学習の様子

学習済みの重み　$w_0 = 0, w_1 = 100, w_2 = 0, w_3 = 0$

分岐履歴

$x_1 = 1, x_2 = -1, x_3 = -1$



- x_4 の方向を予測する
  - x_4 は x_2, x_3 に相関せず，x_1 とのみ相関する場合を考える
    - x_1 に対応する w_1 の重みを大きく，それ以外の絶対値を小さくなるよう学習すればよい
  - 学習：$w_i = w_i + t \times x_i$　（外れた時 or Σ が小さい時）

    （実際の分岐方向 t が 1 の時，x_i が 1 だったなら w_i を大きく
    -1 だったなら w_i を小さく
  - 相関がある x_1 では $t = x_1$ となるため w_1 の絶対値は大きく，
    相関がない x_2, x_3 では t と打ち消しあって w_2, w_3 は平均的に変わらない（x_1 が偏ってない場合）

最初のパーセプトロン予測器の発展形

1. Path-Based 予測器
   - *Jimenez, Daniel A et al., Fast Path-Based Neural Branch Prediction, MICRO 2003*
2. Piecewise Linear 予測器
   - *Jimenez, Daniel A et al., Piecewise Linear Branch Prediction, ISCA 2005*

Path-Based 予測器

- 重みと履歴の内積をパイプライン化
  - ◇ 一度に内積を計算しなくてよくなる
- 動作：
  - ◇ 既存：あるサイクルに重みと履歴の内積を一気に取る
  - ◇ 提案：各サイクルに実行された分岐結果に対応するものを計算

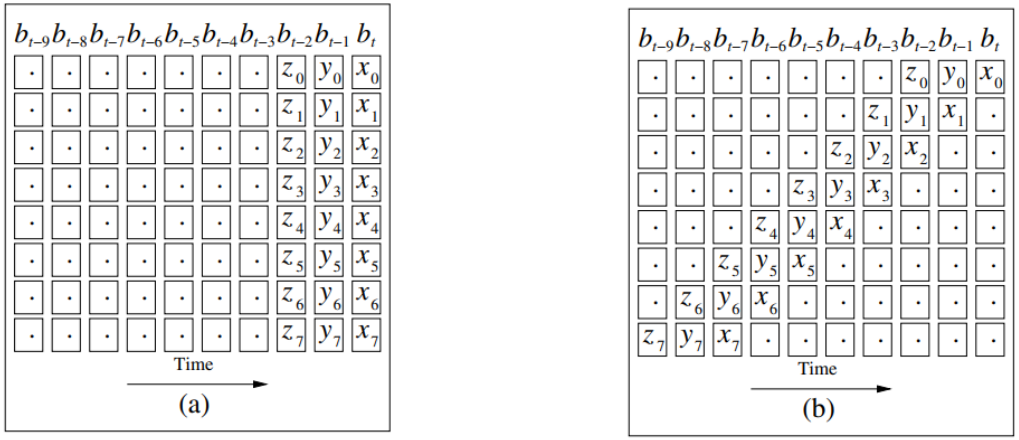


Figure 3. Illustration of the weights used to predict branch b_t with the perceptron predictor (a) and the path-based neural predictor (b) with history length of 7. Vertical columns are weights vectors.

Path-Based 予測器

- 既存：一度に８個を積和
  - ◇ b_t 実行時：$\mathrm{sum} = b_t x_0 + b_{t-1} x_1 + \cdots + b_{t-7} x_7$
- 提案：一度に１個を積和
  - ◇ b_{t-7} 実行時：$sum \ = b_{t-7} x_7$,
  - ◇ b_{t-6} 実行時：$sum \mathrel{+}= b_{t-6} x_6$
  - ◇ b_t 実行時：$sum \mathrel{+}= b_t x_0$

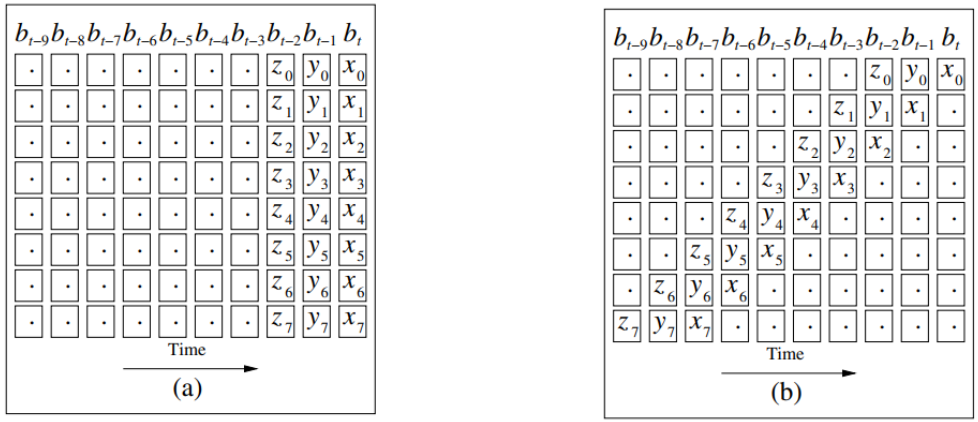


Figure 3. Illustration of the weights used to predict branch b_t with the perceptron predictor (a) and the path-based neural predictor (b) with history length of 7. Vertical columns are weights vectors.

最初のパーセプトロン予測器と Path-Based 予測器の違い

1. 問題：ある分岐からみて将来到達する分岐の重みが全部混じる
   - ◇ b_{t-7} 実行時に b_t の予測に使用する「$sum \ = b_{t-7}x_7$」を計算してしまうが，b_t には複数の分岐が来る
   - ◇ 重み x_7 にそれら全ての学習結果が共有される

2. 利点：どのようなパスを通って自分に至ったかに応じて使う重みが変わる
   - ◇ 既存：b_1 実行時に $b_{t-7}x_7$ 計算すると，x_7 は経路によらず決まる
   - ◇ 提案：b_1 からみて過去に実行された $b_{t-7}x_7$ の x_7 は経路ごとに異なる

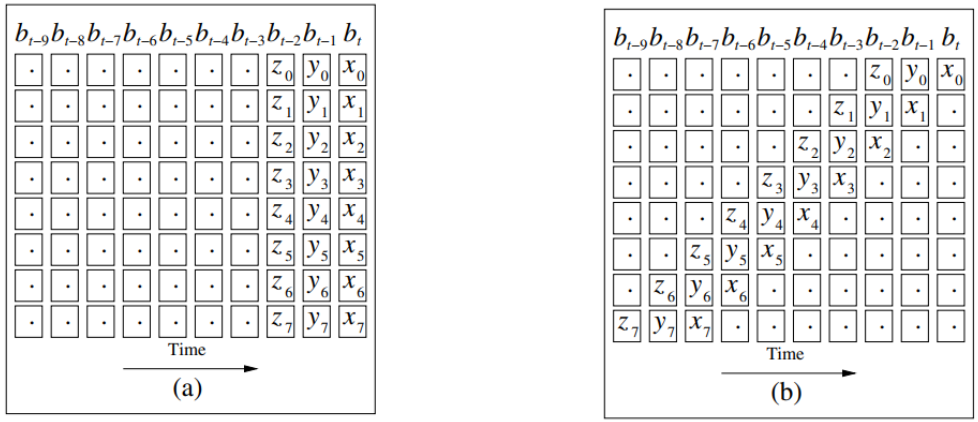


Figure 3. Illustration of the weights used to predict branch b_t with the perceptron predictor (a) and the path-based neural predictor (b) with history length of 7. Vertical columns are weights vectors.

Piecewise Linear 予測器

- Path based 予測器 の特徴
  - ◇ 問題：ある分岐からみて将来到達する分岐の重みが全部混じる
  - ◇ 利点：どのようなパスを通って自分に至ったかに応じて使う重みが変わる
- Piecewise Linear 予測器
  - ◇ 重みが混じらないようにきちんと切り替えて上記問題を解決

b_t の予測に使用する重みテーブルの違い

- Orginal： $w[b_t\_addr][i]$
  - ◇ i 個前に実行した分岐 $b_{(t-i)}$ のアドレスが重みの選択に反映されない
- Path-based： $w[b_{(t-i)}\_addr][i]$
  - ◇ 今予測したい分岐 b_t のアドレスが重みの選択に反映されない

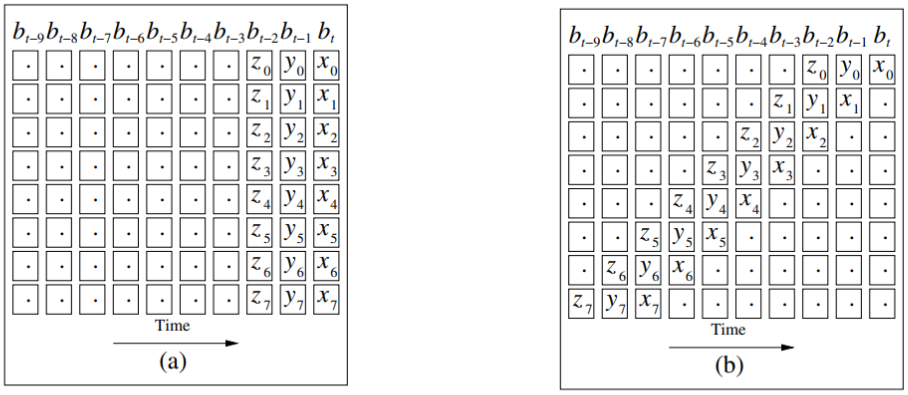


Figure 3. Illustration of the weights used to predict branch b_t with the perceptron predictor (a) and the path-based neural predictor (b) with history length of 7. Vertical columns are weights vectors.

使用する重みテーブルの違い

- Orginal： $w[b_t\_addr][i]$
  - ◇ i 個前に実行した分岐 $b_{(t-i)}$ のアドレスが重みに反映されない
- Path-based： $w\left[b_{(t-i)}\_addr\right][i]$
  - ◇ 今予測したい分岐 b_t のアドレスが重みに反映されない
- 理想 Piecewise Linear： $w[b_t\_addr]\left[b_{(t-i)}\_addr\right][i]$
  - ◇ 重みテーブルを 1 次元ふやせば，どちらも反映できる
- 提案 Piecewise Linear： $w[b_t\_addr \bmod N]\left[b_{(t-i)}\_addr \bmod M\right][i]$
  - ◇ テーブルが大きくなりすぎるので，N と M で mod をとる
  - ◇ N=1 なら Path-based，M=1 なら Original になる
  - ◇ $b_1\_addr \bmod N$ は N 通りの可能性があるので，sum は常に N 並列分やってパイプライン化

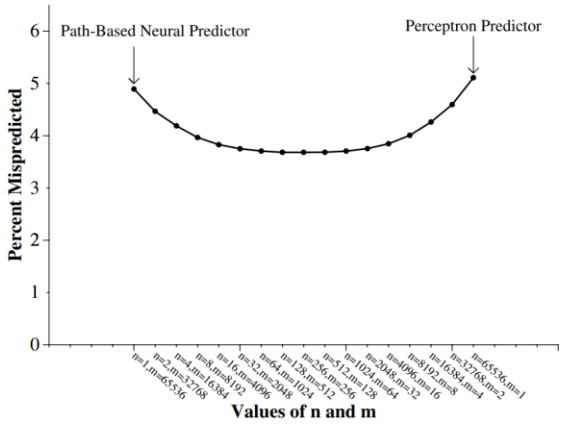
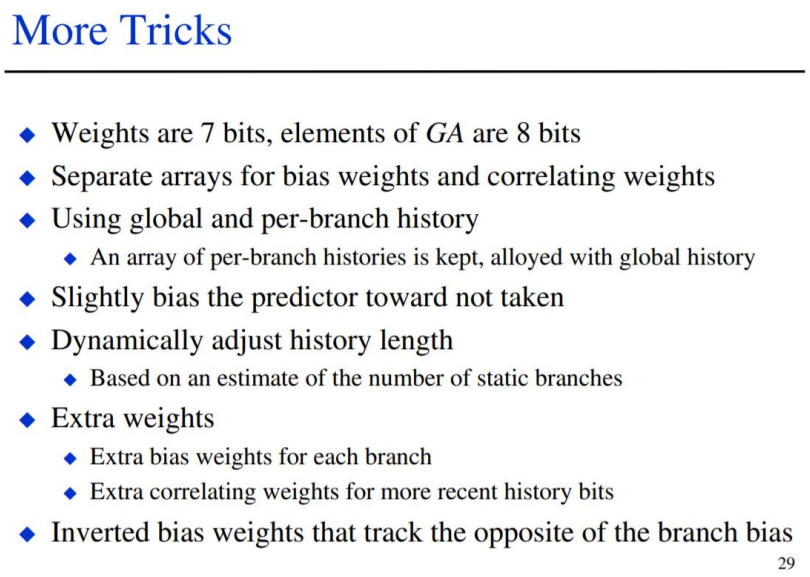


Figure 7. Misprediction rates with values of n

Piecewise Linear の性能

- 冒頭のグラフではそこそこ性能が高い
  - PWL (CBP 版) 8KB： MPKI 5.2 程度
  - TAGE-SC-L 8KB： MPKI 4.2 程度
- ただし，小細工もりもり+相当チューニングされている
  - *Jimenez, Daniel A, Idealized Piecewise Linear Branch Prediction, CBP, 2004*



More Tricks

- Weights are 7 bits, elements of *GA* are 8 bits
- Separate arrays for bias weights and correlating weights
- Using global and per-branch history
  - An array of per-branch histories is kept, alloyed with global history
- Slightly bias the predictor toward not taken
- Dynamically adjust history length
  - Based on an estimate of the number of static branches
- Extra weights
  - Extra bias weights for each branch
  - Extra correlating weights for more recent history bits
- Inverted bias weights that track the opposite of the branch bias

29

分岐予測

1. 方向分岐予測器
    1. 分岐履歴を使った予測器の基本と問題
    2. パーセプトロン予測器
    3. ハッシュ・パーセプトロン予測器
    4. TAGE 予測器
    5. BATAGE 予測器
2. TAGE の実装
3. 間接分岐予測器
4. 複数命令同時フェッチ時の予測の実装方法

ここまでのパーセプトロン予測器は履歴長が固定

- 長い履歴を捉えようとすると履歴長に比例した記憶容量が必要
  - ◇ 下の図の場合，重み行列の行数が増える
  - ◇ 履歴長は数十程度が限界
- しかし，ほとんどの分岐はごく短い履歴長で十分当てられる
  - ◇ 静的分岐予測でも７～８割は当たる

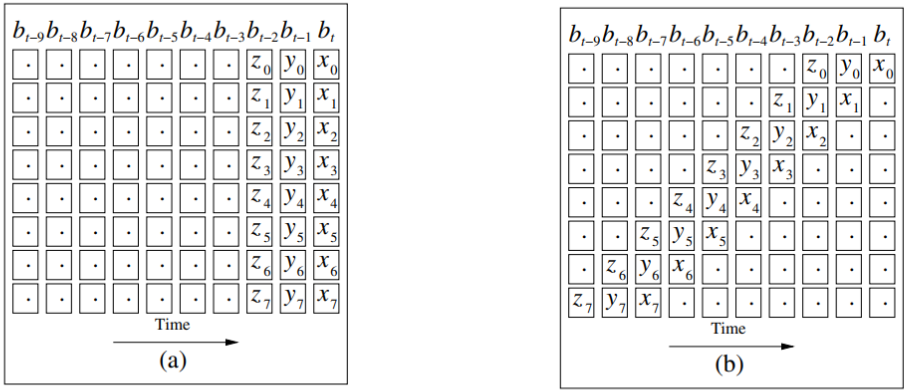


Figure 3. Illustration of the weights used to predict branch b_t with the perceptron predictor (a) and the path-based neural predictor (b) with history length of 7. Vertical columns are weights vectors.

複数履歴長の利用

- 相関する履歴長ごとに異なるテーブルを利用
  - ◇ ハッシュ・パーセプトロン予測器
  - ◇ TAGE 予測器
- 履歴長が長い（数百から1000程度）分岐の予測を可能に

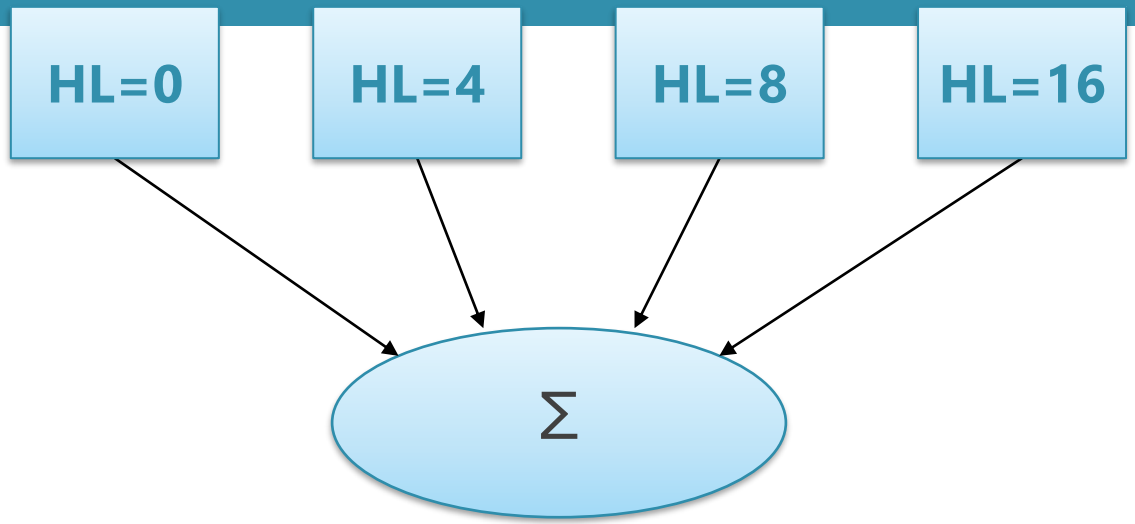
ハッシュ・パーセプトロン予測器

- ハッシュ・パーセプトロン
  - 異なる履歴長を持つ複数のテーブルから重みを読み出し加算
  - （どの論文が最初の提案なのかが，あまりわからない

- O-GEHL：
  - 以下の論文で ハッシュ・パーセプトロン 自体の基本構成が良く説明されている
    A. Seznec. Analysis of the O-GEHL branch predictor, ISCA 2005
  - 基本的なハッシュ・パーセプトロンにテーブル構成の動的調整や学習閾値の動的調整をつけたもの = O-GEHL･･･ だと思う

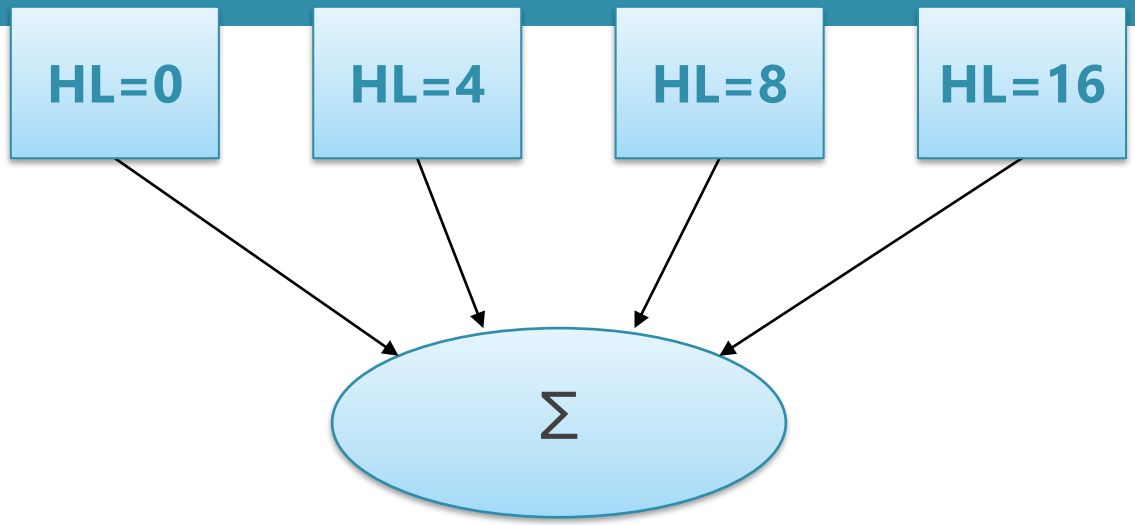
- 実装例
  - https://github.com/ChampSim/ChampSim/blob/master/branch/hashed_perceptron.bpred
  - Data Prefetch Championship 用シミュレータへの実装だが非常にシンプルでわかりやすい

ハッシュ・パーセプトロン



- 履歴長（HL: history length）が異なるテーブルを複数用意
  - ◇ 履歴 と PC の組み合わせでテーブルを引く
  - ◇ 中身は 重み（符号付き 4 ビットなど）にする
- HL=8 のテーブルの場合の例：
  - ◇ 8 ビットの履歴と PC を XOR し，それをインデックスにしてアクセス
    - □ グローバル分岐予測器と同様
  - ◇ 符号付き 4 ビットの重みが取れる

ハッシュ・パーセプトロン



- 全出力を加算してパーセプトロンの処理（ゼロより大きければ taken）
  - 通常のパーセプトロン予測器と違い，各テーブルから出てきた重みはそのまま加算される
    - （重みに何らかの関数を適用するチューニングもある）
- 履歴と重みの内積はとらない
  - 履歴と PC から作ったハッシュ値をワンホットに展開して入力し，テーブルの全エントリで単層パーセプトロン処理を行っていると考えることもできる

ハッシュ・パーセプトロンの効果

- g-share やオリジナル・パーセプトロン予測器の問題
  - 長い履歴を捉えようとすると履歴長に比例した記憶容量が必要
  - 長い履歴を憶えようとすると，短い履歴で済む場合も大きな記憶容量が必要

- ハッシュ・パーセプトロンの効果：
  - 長い履歴をきちんと憶えつつ，履歴が短い大半の場合は少ないエントリ数で記憶できる
    - 相関がある履歴長のテーブルの重みの絶対値が増えるように学習される
    - 相関がない履歴長のテーブルは平均的には変化しない（ランダムに汚される）

ハッシュ・パーセプトロンの効果の例

- たとえば 101 と 11111 の２パターンがあった場合
  （左から右に taken:1, untaken:0）

- 固定長グローバル履歴予測 (HL=5)
  - ５カ所を更新：00101, 01101, 10101, 11101, 11111

- ハッシュ・パーセプトロン (HL=3,5) ：
  - 101：主に１カ所を更新
    - HL=3のテーブルでは，101 の重みを大きく
    - HL=5のテーブルでは，00101, 01101, 10101, 11101 に散らされるので薄く汚す（平均的には変化しない）
  - 11111：主に１ or 2 カ所を更新
    - HL=3のテーブルでは，111 の重みを大きく？
    - HL=5のテーブルでは，11111 の重みを大きく

O-GEHL 予測器

A. Seznec. Analysis of the O-GEHL branch predictor, ISCA 2005

- GEHL：GEometric History Length
  - ◇ 幾何級数的に変化させた HL を使用

- ハッシュ・パーセプトロンからの改良点
  1. 学習の閾値の動的な調整
     - □ 重みの更新は予測ミス時 or Σ が閾値より低い場合に行われる
     - □ この閾値を動的に調整
  2. 履歴長の動的な調整
     - □ ２種類の履歴長の構成を取れる

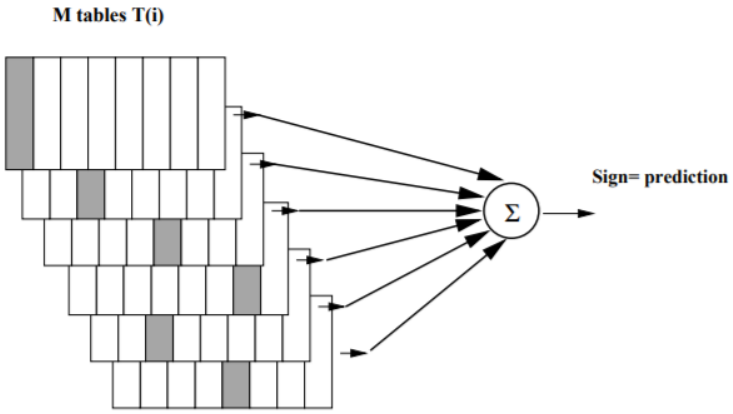
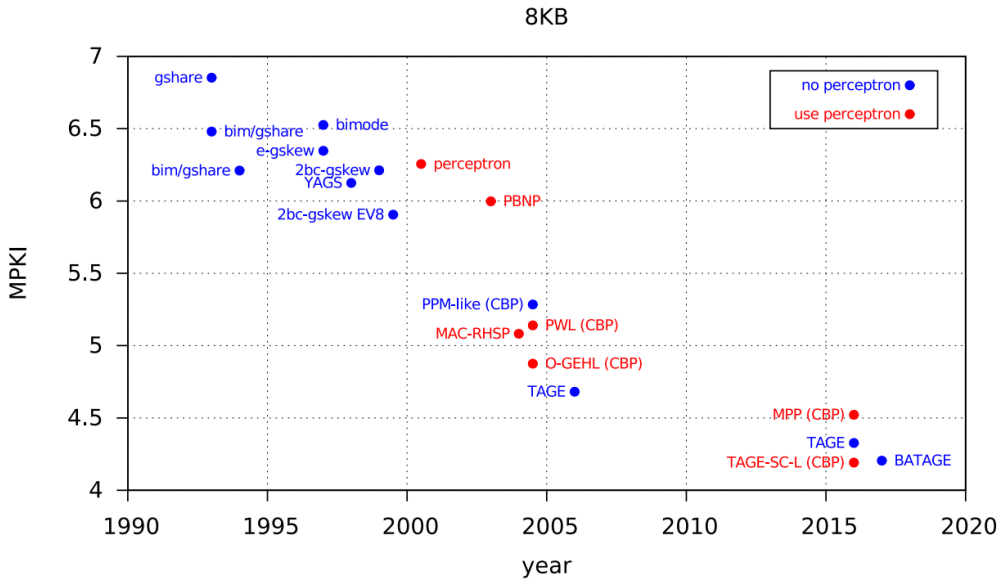


Figure 1. The GEHL predictor

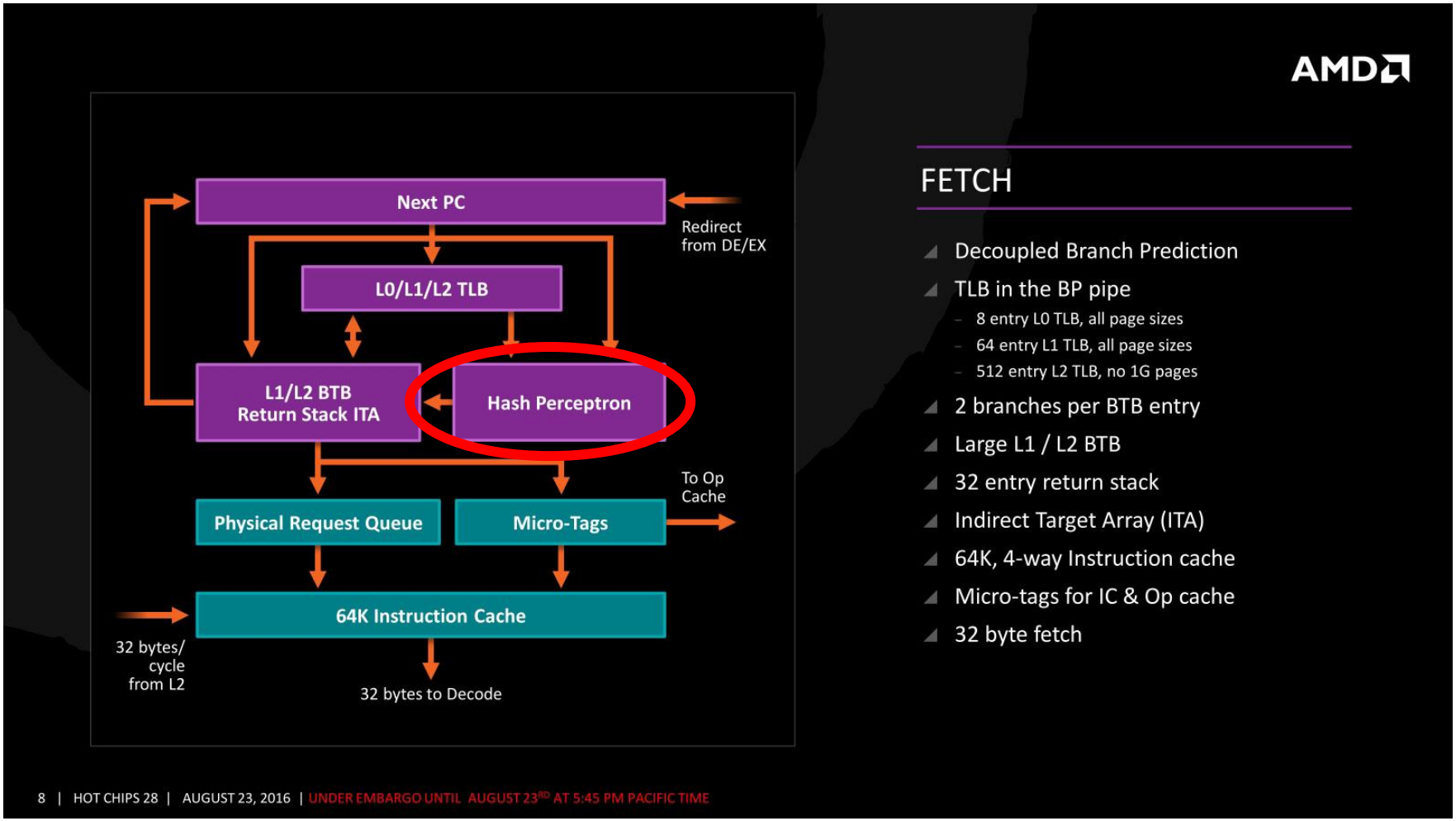
ハッシュパーセプトロンの性能

- Hash perceptron 8KB: MPKI 5.391（未チューニング）
  - ◇ https://github.com/ChampSim/ChampSim/blob/master/branch/hashed_perceptron.bpred
  - ◇ この簡単な実装で小細工もりもり+チューンされた Piecewise Linear (PWL) にも迫る
  - ◇ ここから重みを 8 bit → 4bit にするだけで大分よくなるはず

- O-GEHL はもう一回りよい



AMD Zen ではハッシュ・パーセプトロンを使っている

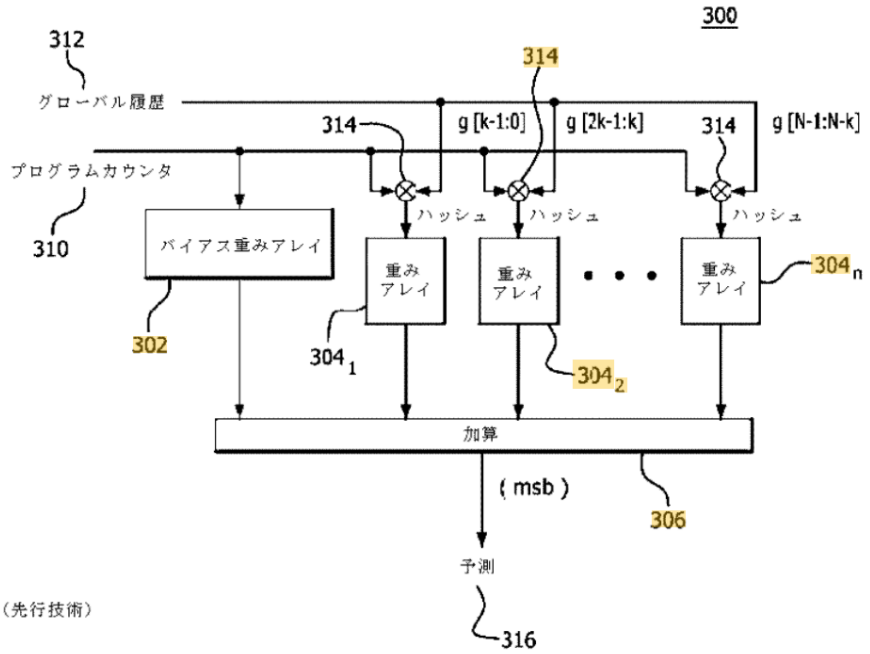


- HotChips 2016 の発表より

AMD の特許の実施例より

- JP6523274B2：
  分岐予測ユニット及びレベル１命令キャッシュにおける帯域幅の増加
  - ◇ https://patents.google.com/patent/JP6523274B2/ja
  - ◇ この特許自体は，フロントエンドの BTB アクセスに関するもの

- 実施例では複数のハッシュ・パーセプトロンを使っている
  - ◇ 高速な LV1 予測器と，低速な LV2 がある想定



（先行技術）

分岐予測

1. 方向分岐予測器
    1. 分岐履歴を使った予測器の基本と問題
    2. パーセプトロン予測器
    3. ハッシュ・パーセプトロン予測器
    4. TAGE 予測器
    5. BATAGE 予測器
2. TAGE の実装
3. 間接分岐予測器
4. 複数命令同時フェッチ時の予測の実装方法

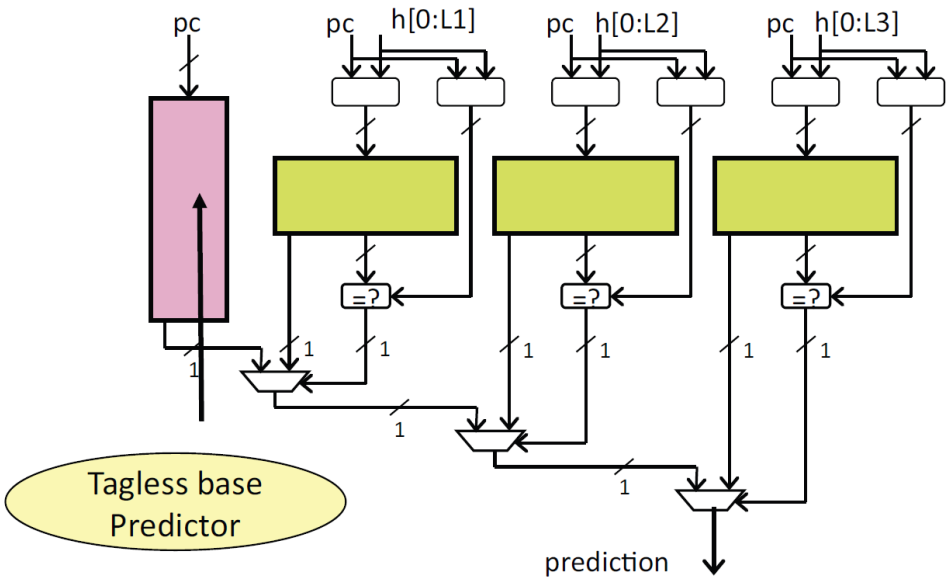
TAGE 予測器

- ひとくちに TAGE と言っても色々ある
- 以降の内容
  1. TAGE 予測器の基本
  2. TAGE 予測器の改良
     1. バンク・インタリービング
     2. 統計的補正
     3. メタ予測

TAGE 予測器

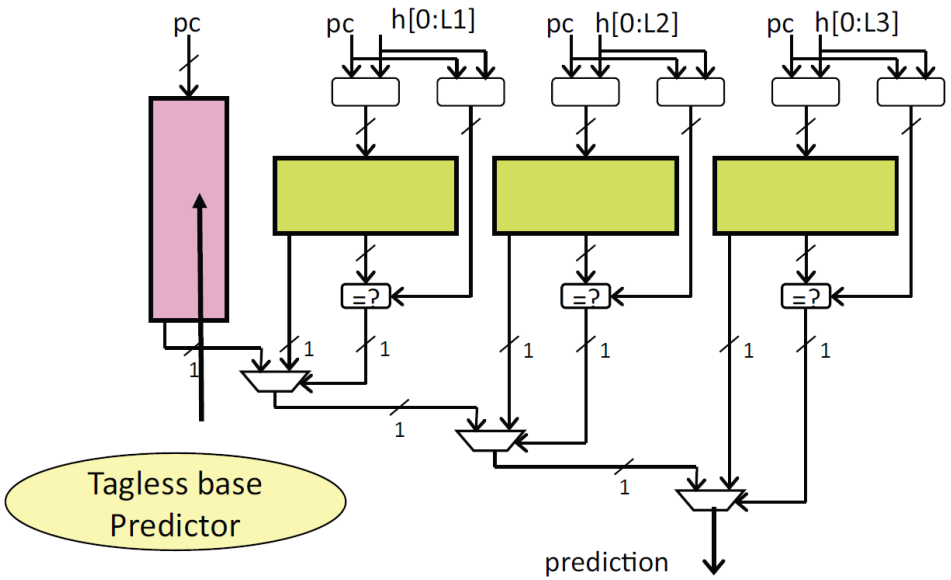
- TAGE 予測器
  - 現在最も予測精度が高いと言われている予測器
  - *A. Seznec and P. Michaud. A case for (partially)-tagged geometric history length predictors, Journal of Instruction Level Parallelism (http://www.jilp.org/vol8), 2006*
- 実機への搭載
  - AMD Zen2 に搭載
  - Intel Haswell は TAGE (8KB)+ITTAGE(13KB) に近い精度を達成しており，搭載されているのではと言われている
    - E. Rohou et al, Branch prediction and the performance of interpreters — Don't trust folklore, CGO 2015
- 構造
  - 異なる履歴長を持つ複数のテーブルからなる
  - 各テーブルはタグを持っており，ヒット・ミスが判定可能

TAGE 予測器



- トーナメント式の構造：
  - 左端：　　単純な２ビット・カウンタ
  - それ以外：
    - グローバル履歴+PC でアクセスしヒット/ミス判定を行う
    - ヒットした場合，そのテーブルの n ビット・カウンタの内容で予測

TAGE 予測器



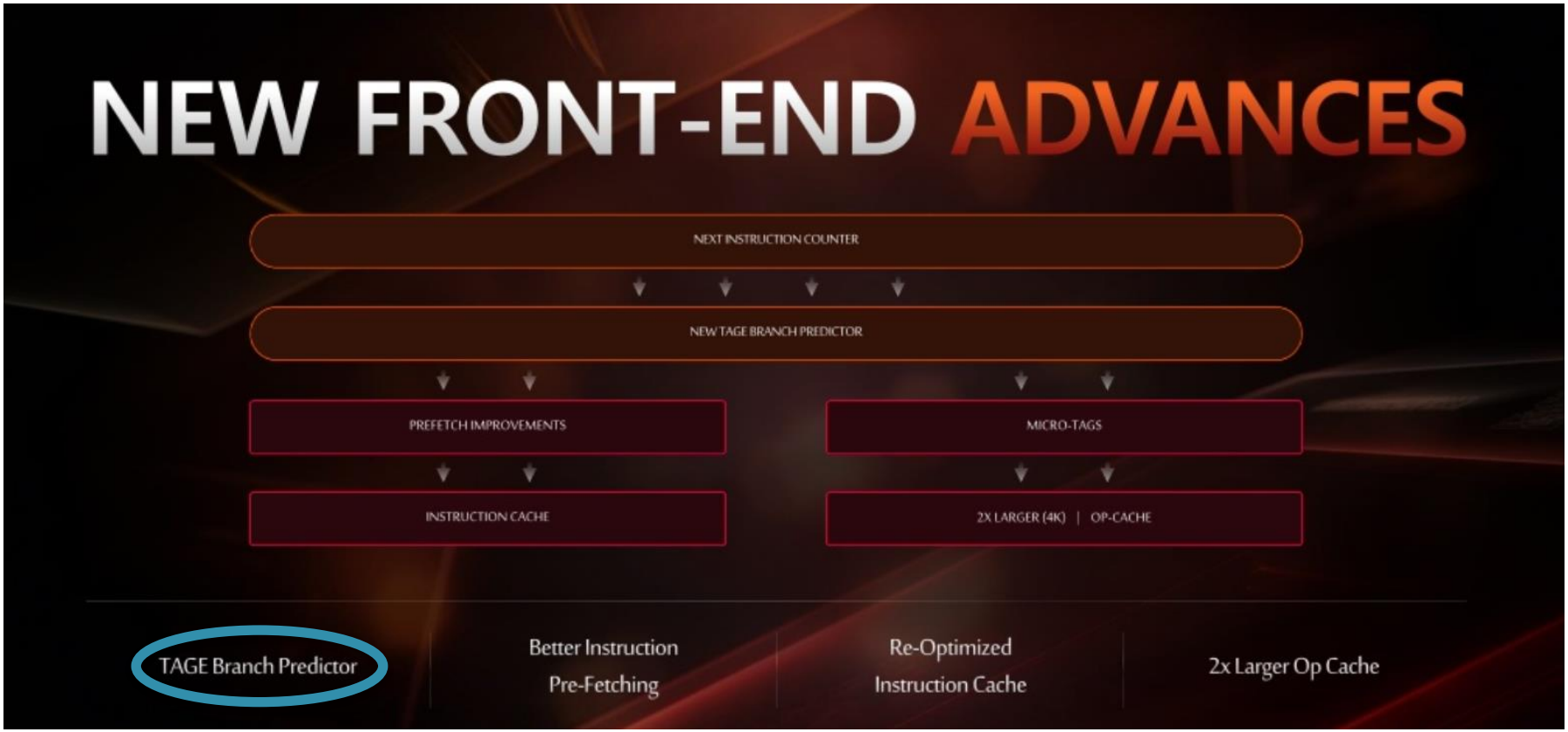
- パターン長が長いテーブルの結果を優先して使う
  - ◇ 右側でヒットするほど，その結果を優先する
  - ◇ 右に行くほど指数的に履歴長が長くなっている
- ヒット/ミスの判定ができるため，トーナメント状に優先度が決定できる
  - ◇ 通常のグローバル履歴予測器やパーセプトロンではできない

TAGE 予測器の利点

- 利点１：パターン長ごとに，最適なテーブルに学習できる
  - ◇ たとえば 101 と 11111 の２パターンがあった場合
    - □ 固定長(5)　　：00101, 01101, 10101, 11101, 11111
    - □ TAGE(3+5)　：101, 11111
- 利点２：履歴長が短い場合の学習が早い
  - ◇ 相関がある履歴長が短い場合に，早く学習が終わる
  - ◇ たとえば静的に方向が決まっている分岐は，2ビット・カウンタが暖まればそれで予測可能
- このあたりの利点はおそらくハッシュ・パーセプトロンにも当てはまる

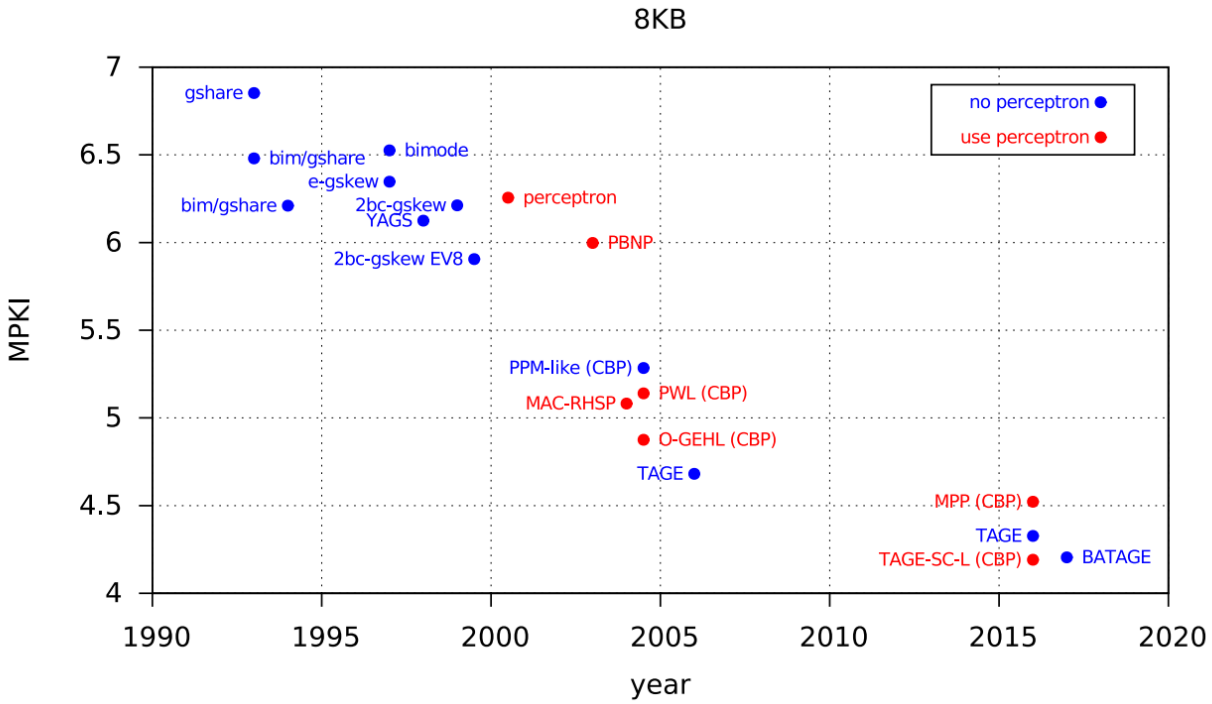
AMD Zen2 (Ryzen 9)

- 方向に分岐予測器に TAGE を導入
  - ◇ 実際は TAGE （低速高精度）とパーセプトロン（高速低精度）のハイブリッドらしい
  - ◇ とりあえずパーセプトロンが言った方向でフェッチしておいて，遅れて TAGE が言った方向が違ったらそっちでやり直す



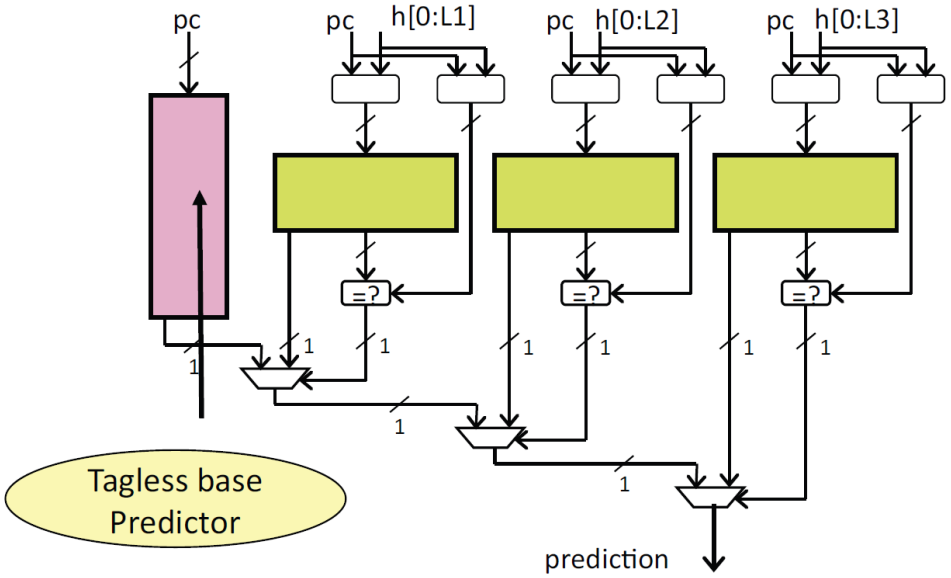
TAGE の発展

- ここまでで説明した TAGE は基本形
  - ◇ 最新のものは１世代分ぐらい改良されている
- TAGE-SC-L (Statistical Corrector + Loop Predictor)
  - ◇ バンク・インターリーブ，統計的補正，メタ予測器，ハッシュ関数の作り方･･･



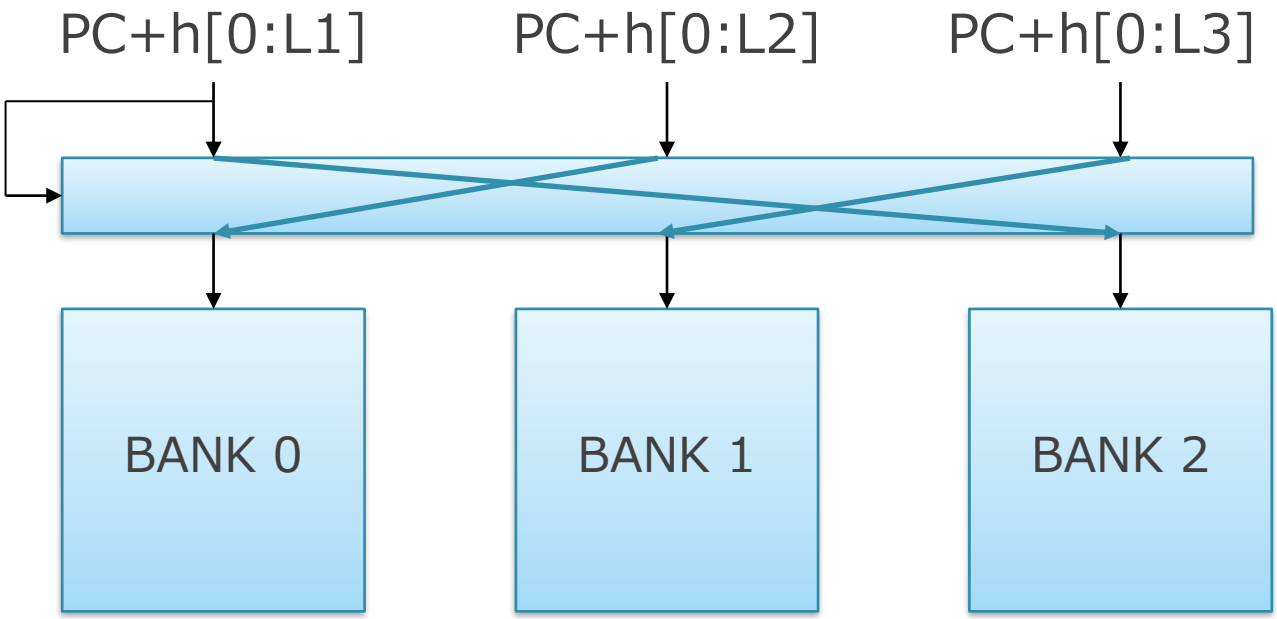
バンク・インターリーブ

- 履歴長の異なる各テーブルのサイズをどうするか？
  - ◇ 短い履歴の方に多くの容量を割り当てる？
- 解決法：バンク・インターリーブ：
  - ◇ マルチバンク化されたテーブルを全履歴長で共有する



バンク・インターリーブ

- 最短の履歴長の 履歴+PC でパーミュテーション
  - 履歴+PC のハッシュ値でそのままマルチバンク化すると，バンク衝突がおきる
  - パーミュテーションにすれば衝突は発生しない
    - クロスバスイッチ等で実装
  - 最短の履歴長の履歴で決定すれば，バンクの行き先は常に一定



バンク・インターリーブ

- 論文にはほとんど書かれていない
  - ◇ しかし，比較的新しい TAGE の実装には入っている
  - ◇ 詳細は CBP に投稿されている実装等を見る必要がある
- かなり効果が高い
  - ◇ TAGE (2005) と TAGE (2015) の差は，これの寄与が大きい
  - ◇ パーセプトロン系との主な性能差になっているとも言える
- コンフリクトの可能性があがるため，タグの幅を少し増やしておく必要がある

Statistical Corrector

- TAGE-SC-L の SC (statistical corrector)
  - *A. Seznec, A New Case for the TAGE Branch Predictor，MICRO 2011*
  - *A. Seznec, TAGE-SC-L Branch Predictors Again, CBP-5 2016*
- 履歴には相関しないが，統計的に方向が偏ってる分岐がある
  - 全く予測はできないが，７割ぐらいは taken など
    - 精度が満たされたら脱出するループでニュートン法を実装
  - テーブルをただ荒らすだけになってしまう
- そう言った分岐を検出して，補正する
  - 補正器はある種のパーセプトロン（GEHL）予測器で構成
  - 論文を読んでもほとんどわからないので，実装をみるしかない

メタ予測器

- TAGE は基本的には最長マッチ･･･
  - ◇ のはずだが，実際には最長(pred)と二番目(altpred)から選ぶ
- これを選択するのが meta predictor
  - ◇ 最長履歴のエントリが確保されたばかりでカウンタが暖まっていない場合に，最長よりも二番目を選ぶ
  - ◇ 最長履歴のカウンタの状態から判定

分岐予測

1. 方向分岐予測器
    1. 分岐履歴を使った予測器の基本と問題
    2. パーセプトロン予測器
    3. ハッシュ・パーセプトロン予測器
    4. TAGE 予測器
    5. BATAGE 予測器
2. TAGE の実装
3. 間接分岐予測器
4. 複数命令同時フェッチ時の予測の実装方法

BATAGE （Bayesian TAGE）

- *Pierre Michaud, An Alternative TAGE-like Conditional Branch Predictor, TACO 2018*
  - ◇ TAGE 自体の説明や問題点，分岐予測器の歴史などまでよくまとまっている（論文誌の論文）
  - ◇ *https://files.inria.fr/pacap/michaud/BATAGE_simulator-1.zip*
- TAGE-SC-L とおよそ同等の性能だが，構造が簡単
  - ◇ メタ予測器や統計補正を不要に

TAGE の問題：Cold counter problem

- Cold counter problem
  - ◇ 暖まっていないカウンタが精度を下げてしまう問題
  - ◇ 統計的な偏りがあると，カウンタが振動してなかなか暖まらない
    - □ カウンタの初期値をどうするか？

- 例：確率 0.1 で taken の場合はすぐ収束するが，0.3 では時間がかかる

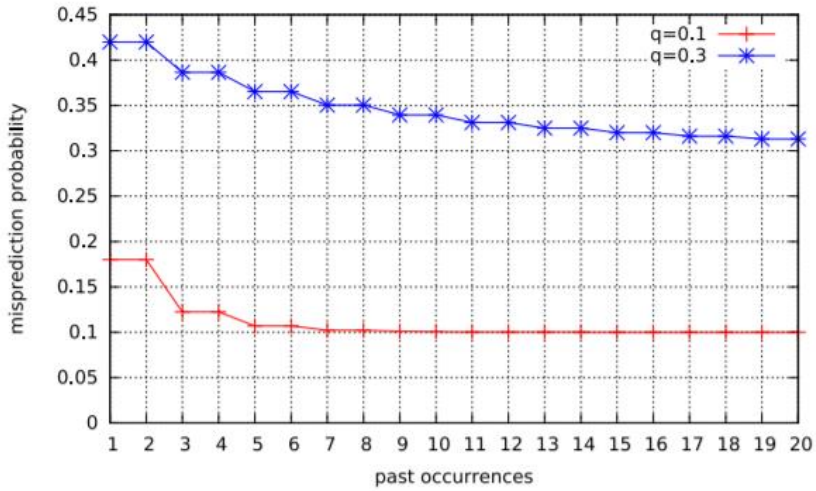


Fig. 3. Illustration of the cold counter problem: misprediction probability for a wide-enough up/down counter on a branch having probability q to be taken, as a function of the number of past occurrences.

BATAGE のアプローチ

- dual counter (n0/n1) の導入
  - taken/not taken をそれぞれ数える
    - 合計値から暖まり具合がわかる
  - TAGE の飽和カウンタでは暖まっているのかどうかわからない
    - up 10/down 8→2 と，up2/down 0→ は見分けが付かない

Fig. 4. Tagged entry in TAGE (left) vs. BATAGE (right). The payload of a TAGE entry consists of an up/down counter (typically, 3 bits) and a u counter (1 or 2 bits). The payload of a BATAGE entry consists of two counters n_1 and n_0 (typically, 3 bits each) counting respectively the number of taken and not-taken occurrences.

Dual counter により統計補正や altpred が不要に

- 暖まり具合が dual counter の合計値から判断できる
  - ◇ 暖まっていない場合は，より暖まっている短い履歴を使う

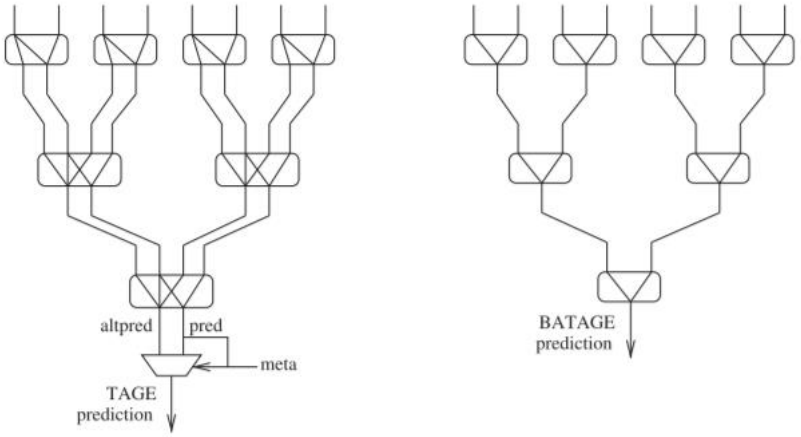


Fig. 5. Divide-and-conquer implementation of the prediction selection circuit for TAGE (left) and BATAGE (right). The circuit for TAGE takes as input the hit/miss bit, the prediction bit, and the low-confidence bit (needed for the final selection between *pred* and *altpred*). The circuit for BATAGE takes as input the $\overline{m}$ value (2 bits) and the prediction bit. This example assumes 7 tagged banks. The path lengths increase from left to right. The leftmost input is the bimodal entry.

置き換え

- TAGE：u bit とグローバルカウンタで制御
  - 自身が正解 + altpred がミスの場合に u bit をセットしてロック
    - そのエントリを守る
  - ミスが全体に増えてきたら u を全体に解除
- BATAGE：確信度が低い場合に行う
  - dual counter を decay させていく
  - アクセスが来ないと下がってくる

Fig. 4. Tagged entry in TAGE (left) vs. BATAGE (right). The payload of a TAGE entry consists of an up/down counter (typically, 3 bits) and a u counter (1 or 2 bits). The payload of a BATAGE entry consists of two counters n_1 and n_0 (typically, 3 bits each) counting respectively the number of taken and not-taken occurrences.

Controlled allocation throttling (CAT)

- 予測困難な分岐では置き換え確率を 1 より下げると精度が上がることが分かっている
  - ◇ 下手にやると，大きく精度が下がる場合がある
  - ◇ 置き換え確率を全体の暖まり具合（確信度）から算出
- CAT を導入しないと，良くチューンされた TAGE に負けるらしい

分岐予測

1. 方向分岐予測器
    1. 分岐履歴を使った予測器の基本と問題
    2. パーセプトロン予測器
    3. ハッシュ・パーセプトロン予測器
    4. TAGE 予測器
    5. BATAGE 予測器
2. TAGE の実装
3. 間接分岐予測器
4. 複数命令同時フェッチ時の予測の実装方法

TAGE の実装：履歴の折りたたみ

- 長い履歴からハッシュ値をどう高速に作るか
  - ◇ リングバッファと XOR を使って折りたたむ（folding）
  - ◇ 当たり前のこととして，この論文の後は明に説明されていない
    - *P. Michaud., A PPM-like ,Tag-Based Branch Predictor, Journal of ILP, Vol. 7, 2005.*

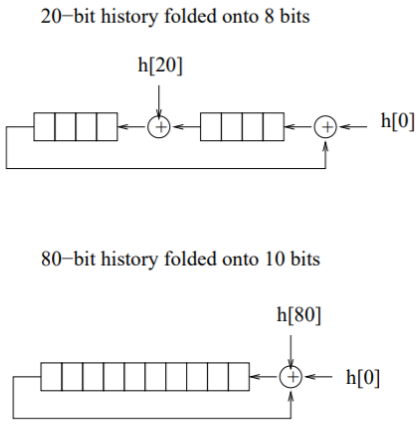


Figure 2: Global history folding can be implemented with a circular shift register (CSR) and a couple of XORs (symbol ⊕).

TAGE の実装：予測ミスからの回復をどうするか

- リングバッファとポインタを使用
  - Global History Register (GHR) をシフトレジスタで作ると大変
  - マルチバンク化されたリングバッファを使う
    - 履歴長ごとにポインタで参照し，ポインタを回復
    - folded history はしょうがないので，保存？
  - 履歴長はバンクコンフリクトが起きないように決める
    - *David J. Schlais et al., BADGR: A practical GHR implementation for TAGE branch predictors, ICCD 2016*

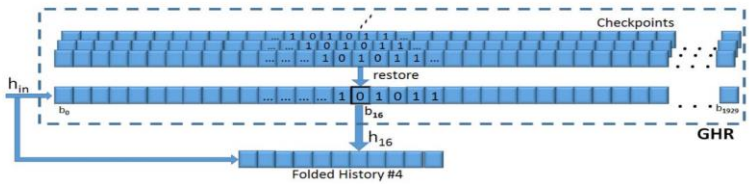


Fig. 4. Fixed-location access: The GHR element, h_n, is always stored at flip-flop (or latch) number b_n. This figure shows traditional checkpointing updating a fourth tagged table folded history.

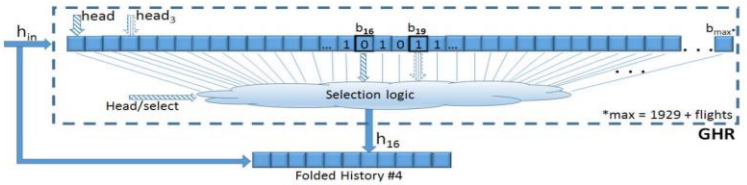


Fig. 5. Variable-location access: The GHR element, h_n, could reside in any of the GHR flip-flops or latches, depending on the head pointer position. This figure shows a circular buffer updating a fourth tagged table folded history. Note the large fan-in to the selection logic.

TAGE 系を実装をする際の注意

- 本来の性能が出ているかわからない
  - ハッシュ関数の作り方（PC からのビットの選択方法）等に，論文では明記されていないノウハウが色々ある･･･らしい
  - スクラッチから自力で実装した場合，著者の実装の精度まで出そうと思うと大変
- CBP-5 の環境にアルゴリズム実装し，公開されている実装との比較や検証を行った方がよい
  - https://www.jilp.org/cbp2016/

分岐予測

1. 方向分岐予測器
    1. 分岐履歴を使った予測器の基本と問題
    2. パーセプトロン予測器
    3. ハッシュ・パーセプトロン予測器
    4. TAGE 予測器
    5. BATAGE 予測器
2. TAGE の実装
3. 間接分岐予測器
4. 複数命令同時フェッチ時の予測の実装方法

間接分岐予測器

- Python のようなインタプリタの性能を大きく改善する
  - Intel プロセッサ（Nehalem, Sandy Bridge, Haswell）と ITTAGE 等の比較
  - *E. Rohou et al., Branch prediction and the performance of interpreters — Don't trust folklore, CGO 2015*
    - ラストオーサーは *A. Seznec*
    - Haswell で相当に改善しているらしい
- 間接分岐予測器
  - ITTAGE (Indirect Target TAGE)
  - BLBP (Bit-level Perceptron Prediction)

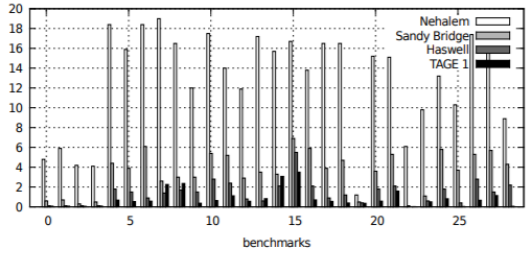


Figure 4. Python MPKI for all predictors

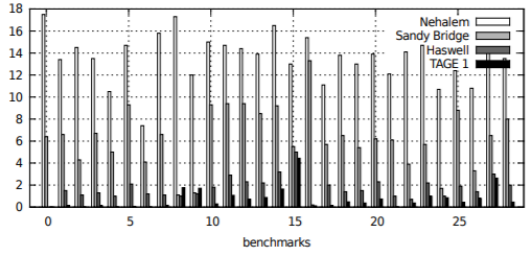


Figure 5. Javascript MPKI for all predictors

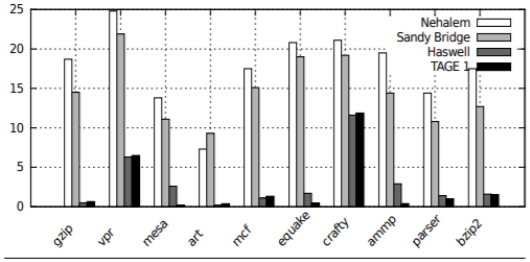


Figure 6. CLI MPKI for all predictors

ITTAGE (Indirect Target TAGE)

- TAGE のカウンタを間接分岐のターゲットに置き換えたと考えて良い
  - *A. Seznec, A 64-Kbytes ITTAGE indirect branch predictor, JWAC-2: Championship Branch Prediction*

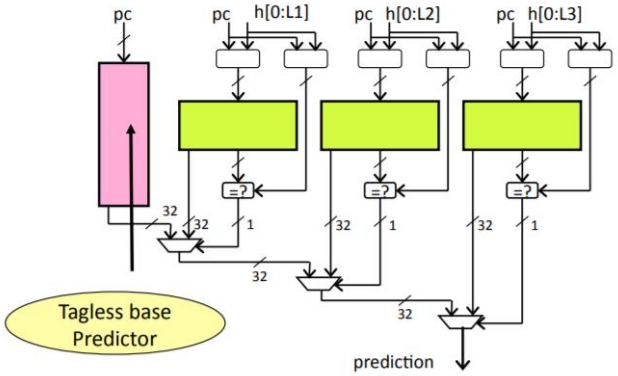


Figure 1: The Indirect Target TAgged GEometric length, ITTAGE, predictor

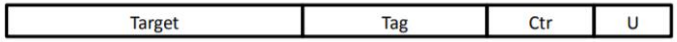


Figure 2: An entry of an ITTAGE predictor table

BLBP (Bit-level Perceptron Prediction)

- *Elba Garza et al., Bit-level Perceptron Prediction for Indirect Branches, ISCA 2019*（ラストオーサーは *Jimenez, Daniel*

- 基本的なアイデア：
  - ◇ ハッシュ・パーセプトロンと同様にしてターゲット・アドレスの各ビットを予測
  - ◇ BTB のセットから複数ターゲットアドレスを読みだす
  - ◇ パーセプトロンの予測結果と最も一致度が高いターゲット・アドレスを選択

- ITTAGE より少しよいぐらい

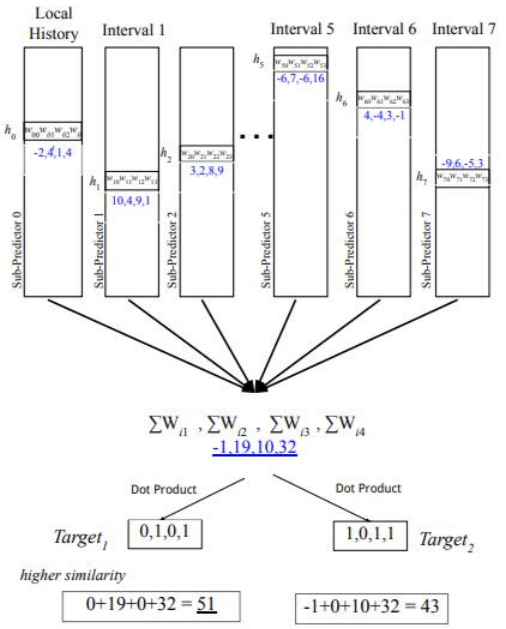


Figure 4: BLBP makes a prediction by aggregating the likelihood for two example targets from eight sub-predictors. Each sub-predictor is trained with different history lengths or local history.

分岐予測

1. 方向分岐予測器
    1. 分岐履歴を使った予測器の基本と問題
    2. パーセプトロン予測器
    3. ハッシュ・パーセプトロン予測器
    4. TAGE 予測器
    5. BATAGE 予測器
2. TAGE の実装
3. 間接分岐予測器
4. 複数命令同時フェッチ時の予測の実装方法

複数命令フェッチへの対応

- 複数命令を同時にフェッチする場合の分岐予測の問題
  - ◇ フェッチ・グループ内のどこに分岐があるのかわからない
- たとえば,
  - ◇ PC+0, PC+1, PC+2, PC+3 の４つを同時にフェッチ
  - ◇ これのどれが分岐なのか？
  - ◇ 方向予測器や BTB はどの PC で引くべきか？

ナイーブな方法と問題

- フェッチ・グループの先頭で引く
  - PC+0 で方向予測器と BTB をアクセス
  - そのグループ内にある最初の分岐の予測を行うと考える
  - PC+2 に分岐があるとすると，PC+0 で PC+2 の予測を行う
- 分岐が連続して存在すると悲惨なことに
  - グループ内に複数分岐があると対応出来ない
    - PC+3 に分岐があると対応できない
    - PC+2 でフェッチを止める必要がある
  - グローバル履歴がずれる
    - untaken な分岐は BTB に登録されないので，グローバル履歴にも入らない
    - ある日 taken になると，履歴がそこからずれる

Basic block BTB

- *Kumar, R, Boomerang: a Metadata-Free Architecture for Control Flow Delivery, HPCA 2017*
  - この論文自体は，分岐予測を run ahead させて i-cache と BTB をプリフェッチするアイデアを提案

- Basic block BTB
  - Basic block 単位で
    ターゲット・アドレスを入れる
  - 先頭の PC で引くと，最初の
    分岐までの距離がわかる

- 問題：
  - ずっと untaken の分岐についても
    エントリを消費してしまう

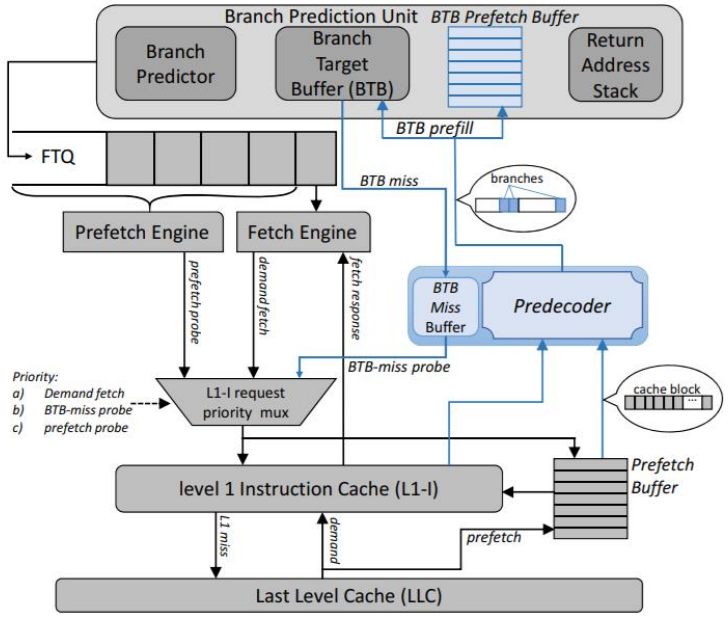


Figure 6. Boomerang microarchitecture. New components in light blue. (FTQ: Fetch Target Queue.)

ARM の実装方法

- *Yasuo Ishii et al., Re-establishing Fetch-Directed Instruction Prefetching: An Industry Perspective, ISPASS 2021*
  - ◇ ARM の石井さんが「アカデミックの連中に本物のフロントエンドを教えてやる」とかいって執筆
  - ◇ 本題は，大きな BTB と decoupled frontend (fetch directed prefetching) で単純かつかなり高い性能がでるというもの
- 複数命令同時フェッチの問題を target history + マルチバンク化により解決

Target history

$$direction\_history = (direction\_history \ll 1) \mid branch\_taken \quad (1)$$

$$target\_hash = (instruction\_address \gg 2) \oplus (target \gg 3) \quad (2)$$

$$target\_history = (target\_history \ll 2) \oplus target\_hash \quad (3)$$

- 履歴の取り方の違い
  - 通常の履歴：分岐方向を 1/0 で表現
  - target history：taken 時の命令のアドレスと target を使う
- 利点：Untaken の間は history が変化しない
  - 通常の履歴の場合：
    - PC+1 は，PC+0 が分岐かどうかで履歴が変化する（分岐なら 0 が挿入される）が，予測時にはわからない
  - Target history の場合：
    - untaken の間は history は変化しないので，PC+1 は PC+0 と同じヒストリを使える

Target history + 予測器のマルチバンク化

- フェッチ・グループ内の複数の分岐を同時に予測する
- 方向予測器のテーブルのアクセス
  - target history を PC の上位と XOR
  - PC+0, PC+1, PC+2・・・で下位ビットはかぶらないので，コンフリクトなしにマルチバンク化可能
  - 方向の履歴では，分岐の有無で 2 番目以降のインデクスが変わるためこれができない
- BTB はマルチバンク化+セット位置の計算方法を変える
  - セットを決める際のインデクスを命令単位ではなく，16B 単位にする
  - 同一セットに同一フェッチ・グループのターゲットが複数入る

Target history を使った実装の備考

- TAGE やハッシュ・パーセプトロンにもそのまま使える
  - GHR の幅が 1 bit から 6 bit 程度（target hash）の幅になる
  - ヒストリの持つ情報量が増えるので，精度もあがる
    - BATAGE の実装では使われている
  - バンクが増えすぎるので，これらをさらにマルチバンク化するのはしんどそう

- 方向分岐予測のマルチバンク化は，階層予測器の LV2 以降がよい？
  - LV1 予測器（１サイクルで予測）は，g-share のような簡単なものにし，マルチバンク化
  - LV2 予測器は BTB のヒットミス判定後に，分岐位置が確定したらその PC で LV2 予測を開始

Decoupled front-end

- 分岐予測器と i-cache は非同期に動く
  - 分岐予測器は i-cache より先行してアドレスを生成
  - ヒット・ミス確認をしてミスなら L2 にリクエスト
    - L2 から届くのを待たずに後続の処理を続行
  - 命令の読み出しは in-order にバッファないしはキャッシュから行う

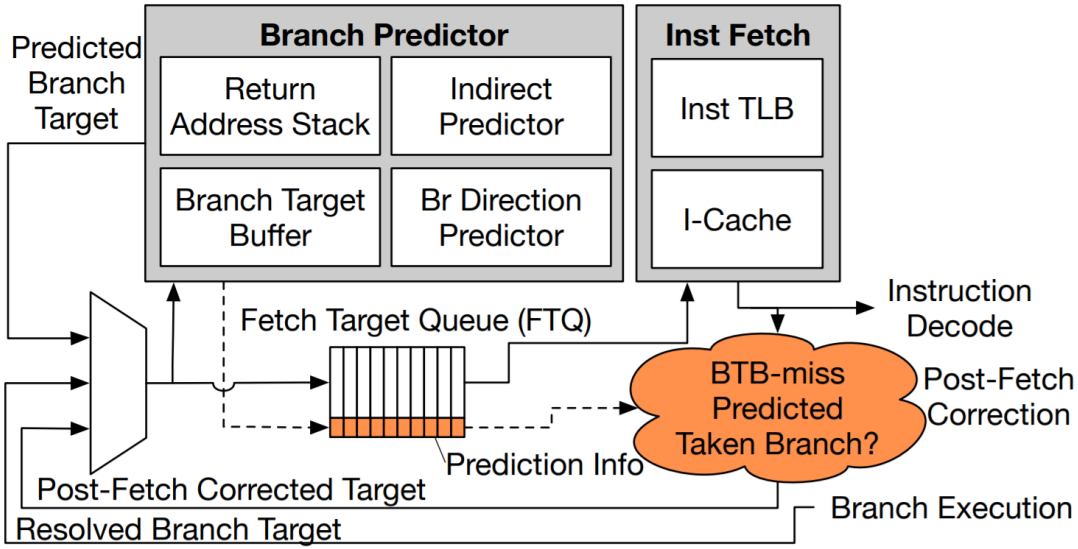


Fig. 4. Overview of frontend microarchitecture design.

実際の実装例

- *Andrea Pellegrini et al., The Arm Neoverse N1 Platform: Building Blocks for the Next-Gen Cloudto-Edge Infrastructure SoC, IEEE MICRO 2020*
  - ARM N1 が石井さん（この論文では共著者）の前述の論文の実装になっていることがうかがえる
- *Adam Collura et al., The IBM z15 High Frequency Mainframe Branch Predictor, ISCA 2020*
  - 汎用機むけの分岐予測器の実装
  - 実際の製品の中身について書かれたものとしては，驚異的に詳しい
    - TAGE やパーセプトロンも実際に使われている模様
  - Basic block BTB も使っている？
- JP6523274B2：https://patents.google.com/patent/JP6523274B2/ja
  分岐予測ユニット及びレベル１命令キャッシュにおける帯域幅の増加
  - AMD の特許．高速に LV1 分岐予測を回すための方法が提案．

分岐予測のまとめ

- 各種の分岐予測器，間接分岐予測器，フロントエンド構成方法についての研究を紹介
- 特に以下の論文や実装が参考になる
  - Champsim の Hash perceptron
  - BATAGE の論文や実装
  - 石井さんの ISPASSの論文
  - IBM z15 の分岐予測器の論文
- 予測器の検証では，CBP-5 環境でテストすると良いと思う