社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2006－EMB－1 (1)
2006／6／27

分岐予測器の最悪フラッシュタイミングの効率的解析手法

小西昌裕[†,☆] 中島浩[†] 中田尚[†]
津邑公暁[†,☆☆] 高田広章[††]

本論文は，与えられたプログラムの実行中に生ずる F 回の割り込みが，どのようなタイミングで生じると2ビットカウンタによる分岐予測ミスが最大となるかを解析する効率的なアルゴリズムについて述べる．プログラムが実行する条件分岐の数を N としたとき，まずこの問題が動的計画法により $O(N^2F)$ の時間計算量で解析できることを示す．続いて実用的なプログラムについて，実効的に $O(NF)$ の時間計算量で解析するアルゴリズムを示し，SPEC CPU95 ベンチマークを対象とした評価により，このアルゴリズムの効率性を実証する．

An Efficient Analysis of Worst Case Flush Timings for Branch Predictors

Masahiro Konishi,[†,☆] Hiroshi Nakashima,[†] Takashi Nakada,[†]
Tomoaki Tsumura[†,☆☆] and Hiroaki Takada[††]

This paper proposes an efficient algorithm to find the *worst case flush timings* for a given program with respect to the number of branch mispredictions. First we give a basic algorithm based on dynamic programming which takes $O(N^2F)$ computation time for a program with N conditional branches and F flush timings. Then we show its improvement to achieve $O(NF)$ time approximately for practical programs with its proof obtained by our evaluation with SPEC CPU95 benchmarks.

1. Introduction

リアルタイムシステムのプログラム設計において，プログラムが所与の時間制約までに実行を完了することを検証・保証するための解析の一つに，**最悪フラッシュタイミング** (**WCFT**: Worst Case Flush Timing) の解析がある．これは，あるプログラムの実行中に（高々）F 回のプリエンプションが生じるとしたとき，その影響でプログラムの性能がどの程度劣化するかを見積もるものである．たとえば宮本らは，性能への影響が大きいキャッシュの WCFT 解析をメモリアクセスログに基づいて行い，N 回のメモリアクセスに対して $O(N^2F)$ の時間計算量でキャッシュミス回数を最大化するタイミングを求められることを示すとともに，実用的プログラムに対して時間計算量が実効的に $O(NF)$ となるアルゴリズムを示している[1)]．

一方，多くのマイクロプロセッサには，キャッシュと同等の重要度を持つ高速化機構として分岐予測器が備えられているが，そのプリエンプションに関する最悪挙動についてはほとんど検討されていない．その理由として，分岐予測器とプリエンプションの関係がキャッシュよりも複雑であることが挙げられる．たとえば，プログラム実行中の任意の時点におけるキャッシュの最悪状態は，全てのブロックが無効化された状態として容易に定義できるが，2ビットカウンタ予測器[3)] にはこのようなわかりやすい最悪状態は存在せず，ある時点での最悪状態はそれ以後の全ての分岐命令の挙動に左右される．またある時点での予測器のカウンタ値は，それ以前に実行された不定個の分岐命令の挙動によって定まるため，連想度に等しい数の異なるブロックへのアクセスにより状態が決定するキャッシュに比べて，過去のアクセス履歴との依存性が解消されにくいという問題もある．

本論文は，この分岐予測器に関する WCFT 解析という困難な問題に対して，以下の2点で貢献するものである．まず最初に，分岐予測器についてもキャッシュと同様に，動的計画法を用いることによって $O(N^2F)$

† 豊橋技術科学大学
Toyohashi University of Technology
†† 名古屋大学
Nagoya University
☆ 現在，(株)PFU
Presently with PFU Limited
☆☆ 現在，名古屋工業大学
Presently with Nagoya Institute of Technology

の時間計算量で解析できることを示す．第2の貢献はより重要なものであり，動的計画法アルゴリズムを改良することによって実効的に時間計算量が$O(NF)$に改善できることを示すとともに，それが実用的プログラムに対して成り立つことをSPEC CPU95ベンチマークを用いた評価によって実証する．なお，本論文で示すアルゴリズムの詳細とそれらの正当性の証明については，文献[2]を参照されたい．

2. 定義と表記

まず最初に，本論文で解析対象とする分岐予測器を定義する．

定義1 解析対象のワークロードにより実行されたN個の条件分岐命令を，その実行順に$b_1, \ldots, b_N$とし，Pを2ビット飽和カウンタ$c[0], \ldots, c[|P|-1]$の集合とし，また$e(i)$を分岐b_iが参照するカウンタ$c[e(i)]$を与える関数とする．分岐b_iは$c[e(i)] \geq 2$（または$c[e(i)] < 2$）であるとき**成立**（あるいは**不成立**）と予測される．またb_iが実際に成立（あるいは不成立）であることを$b_i \mapsto \mathrm{T}$（あるいは$b_i \mapsto \mathrm{N}$）と表記し，そのとき$c[e(i)]$は$\min(c[e(i)]+1, 3)$（あるいは$\max(c[e(i)]-1, 0)$）に「アトミック」に更新される．成立（あるいは不成立）の分岐b_iの予測は，$c[e(i)] < 2$（あるいは$c[e(i)] \geq 2$）であったときに**失敗**したという． □

次に**最悪フラッシュタイミング**（**WCFT**）と，それに関連するコストを定義する．

定義2 $m_e^\gamma(i,j)$を，分岐b_iの実行直後（あるいは$i=0$であればプログラム実行開始時）に$c[e]=\gamma$である場合に，集合$\{b_k \mid i < k \leq j,\ e(k)=e\}$中の分岐の中で予測が失敗したものの総数とする．プログラムがb_iを実行してからb_{i+1}を実行するまで（$i=0$であればb_1を実行するまで）のいずれかの時点で割り込まれることをiでの割り込みと呼び，その結果$c[e]$が任意の値に**フラッシュ**された場合，$m_e^\gamma(i,j)$の最悪値は$m_e(i,j) = \max_{0 \leq \gamma \leq 3}\{m_e^\gamma(i,j)\}$で与えられる．したがって$i$での割り込みによって$b_{i+1}, \ldots, b_j$が被る**フラッシュコスト**は，$C(i,j) = \sum_e m_e(i,j)$と定義できる．$i$での割り込みと，それに続く$f$回の割り込みによる最悪のフラッシュコストは，下式によって定義される．

$$\Gamma(i,f) = \max\left\{ \sum_{k=0}^{f} C(j_k, j_{k+1}) \;\middle|\; i = j_0 \leq j_1 \leq \cdots \leq j_f \leq j_{f+1} = N \right\} \quad (1)$$

したがって$\Gamma(0,F)$は，$j_1, \ldots, j_F$でのF回の割り込みによる最悪コストを与える． □

3. WCFTアルゴリズム

3.1 動的計画法によるアルゴリズム

宮本らによる定式化[1]と同様に，$C(j_k, jk+1)$が他の$C(j_{k'}, j_{k'+1})$とは完全に独立に定まることから，式(1)は以下のように変形できる．

$$\Gamma(i,f) = \max_{i \leq j \leq N}\{C(i,j) + \Gamma(j, f-1)\} \quad (2)$$

この漸化式と，式(1)から定まる初期条件$\Gamma(i,0) = C(i,N)$により，以下の動的計画法に基づくアルゴリズム*DPWCFT*が得られる．

Algorithm DPWCFT:
$\Gamma[0 \ldots N][0] \leftarrow cost(N)$;
for $f = 1$ **to** F **do begin**
  $\Gamma[0 \ldots N][f] \leftarrow 0$;
  for $j = N$ **downto** 0 **do begin**
    $C[0 \ldots j] \leftarrow cost(j)$;
    for $i = j$ **downto** 0 **do begin**
      $\Gamma[i][f] \leftarrow \max\{C[i] + \Gamma[j][f-1],\ \Gamma[i][f]\}$;
    end
  end
end

なお上記において，$cost(j)$は$0 \leq i \leq j$なる全てのiについての$C(i,j)$を，後述の方法で求める関数である．したがって，このアルゴリズムが終了した時点で$\Gamma[0][F]$の値は$\Gamma(0,F)$となる☆．後述のように$cost(j)$の時間計算量は$O(j)$であるので，上記のアルゴリズムの時間計算量は$O(N^2F)$となることは明らかである．空間計算量は，最悪コストだけでなくそれをもたらすタイミングが必要であれば$O(NF)$となるが☆☆，最悪コストだけが必要な場合は$\Gamma[0 \ldots N][f]$と$\Gamma[0 \ldots N][f-1]$のみを保持すれば十分であるので$O(N)$となる．

次に$cost(j)$の時間計算量が$O(j)$であることを示す．$p(k)$をb_kの直前にカウンタ$c[e(k)]$を参照する分岐の番号，すなわち$p(k) = \max\{l < k \mid e(l) = e(k)\}$とし，そのような分岐が存在しなければ$p(k) = 0$とする．ここでたとえば$\gamma = 0$と$\gamma = 1$について$m_{e(k)}^\gamma(k,j)$

☆ $i > 0$に関する$\Gamma(i,F)$は不要であるので，最後の繰り返し$f = F$は$\max_j\{C(0,j) + \Gamma(j, F-1)\}$を計算するように修正することができ，実際にそのように実装している

☆☆ 全てのiとfに対して$C(i,j) + \Gamma(j, f-1)$を最大化するjを保持することは容易であるが，上記のアルゴリズムでは簡潔な記述とするために省略している．

が既知であるとすると，$p(k) \le k' < k$ なる全ての k' に関する $m^0_{e(k)}(k', j)$ は，下式によって求めることができる．

$$m^0_{e(k)}(k', j) = \begin{cases} m^1_{e(k)}(k, j) + 1 & b_k \mapsto \mathrm{T} \\ m^0_{e(k)}(k, j) & b_k \mapsto \mathrm{N} \end{cases}$$

すなわち，$c[e(k)]$ が $b_{p(k)}$ と b_k の間のいずれかの時点で 0 となったとし，かつ b_k が成立したとすると，b_k の予測は失敗して $c[e(k)]$ は 1 となるので，予測失敗の総数は $m^1_{e(k)}(k, j) + 1$ となる．一方 b_k が不成立であれば，その予測は成功して $c[e(k)]$ の値は 0 のままとなる．同様の漸化式は $\gamma \ge 1$ についても容易に定まり，また任意の e, γ および j について $m^\gamma_e(j, j) = 0$ であることはその定義により定まるため，全ての γ と $0 \le i \le j$ なる i について $m^\gamma_e(i, j)$ を $O(j)$ で求めるアルゴリズムはほぼ自明である．したがって，$m^\gamma_e(i, j)$ の最大値である $m_e(i, j)$ も容易に $O(j)$ で求まり，さらに定義 2 から得られる漸化式

$$C(i-1, j) = C(i, j) + (m_{e(i)}(i-1, j) - m_{e(i)}(i, j))$$

を用いれば全ての i に関する $C(i, j)$ も $O(j)$ で求めることができる．

3.2 飽和分岐列

N は一般に非常に大きく，たとえば SPEC CPU95 ベンチマークであってもその大半で 1 億を越えるため，$O(N^2F)$ のアルゴリズムは実用的であるとはいえない．したがって *DPWCFT*() を改善し，少なくとも実用的プログラムに対して $O(NF)$ の時間計算量とする必要がある．この改善のためには，*cost*(j) と *DPWCFT*() の最内ループの時間計算量 $O(j)$ を $O(1)$ とする必要がある．

改善のキーとなるアイデアは，分岐の成否が一定のパターンとなる分岐列は，それらが参照するカウンタを初期値によらず一定の値に収束させるという事実に基づく．たとえば特定のカウンタを参照し全て成立する 3 つの分岐（TTT と表記）は，それらが実行された後のカウンタ値を確実に 3 にする．一般に TT(NT)*T なる分岐列はカウンタを確実に 3 にし，NN(TN)*N なる列は確実に 0 にする．このような列を**飽和分岐列(SBS**: Saturating Branch Sequence) と呼ぶ．全ての SBS は，TT(NT)* + NN(TN)*N なる正規表現を受理する有限オートマトンによって，明らかに $O(N)$ の時間計算量によって検出することができる．

アルゴリズムの性能改善に，SBS は 2 つの役割を果たす．b_h と b_t を，カウンタ $c[e]$ を参照する（すなわち $e = e(h) = e(t)$）SBS の先頭と末尾とすると，$c[e]$ は $b_h, \ldots, b_t$ の実行後には必ず 0 または 3 となるので，全ての $i < h$ と全ての $j \ge t$ に関する $m_e(i, j)$ は，以下によって求めることができる．

$$\vec{m}_e(t, j) = \begin{cases} m^0_e(t, j) & b_t \mapsto \mathrm{N} \\ m^3_e(t, j) & b_t \mapsto \mathrm{T} \end{cases}$$

$$m_e(i, j) = m_e(i, t) + \vec{m}_e(t, j) \qquad (3)$$

ここで $\overleftarrow{h}(j)$ と $\overleftarrow{t}(j)$ を $e(j)$ に関する SBS の中で b_j に先行する直近のものの先頭と末尾とし，同様に $\vec{h}(i)$ と $\vec{t}(i)$ を $e(i)$ に関する SBS の中で b_i に後続する直近のものの先頭と末尾とする．このとき上式 (3) により，$C(i, j)$ に関する 2 つの漸化式を得ることができる．

$$\begin{aligned} &e = e(j) \wedge t = \overleftarrow{t}(j) \Rightarrow \forall i < \overleftarrow{h}(j): \\ &\quad C(i, j-1) = C(i, j) - (m_e(i, j) - m_e(i, j-1)) \\ &\qquad\qquad = C(i, j) - (\vec{m}_e(t, j) - \vec{m}_e(t, j-1)) \\ &\qquad\qquad \equiv C(i, j) - \vec{\delta}(j) \qquad (4) \\ &e = e(i) \wedge t = \vec{t}(i) \Rightarrow \forall j \ge \vec{t}(i): \\ &\quad C(i-1, j) = C(i, j) + (m_e(i-1, j) - m_e(i, j)) \\ &\qquad\qquad = C(i, j) + (m_e(i-1, t) - m_e(i, t)) \\ &\qquad\qquad \equiv C(i, j) + \overleftarrow{\delta}(i) \qquad (5) \end{aligned}$$

ここで式 (4) は，全ての $i < \overleftarrow{h}(j)$ に関する $C(i, j-1)$ を，$C(i, j)$ と i に無関係に定まる $\vec{\delta}(j)$ から求めることができ，*cost*() による繰り返し計算が不要であることを示唆している．また式 (5) より，全ての $j \ge \vec{t}(i)$ に関する $C(i-1, j)$ を，$C(i, j)$ と j に無関係に定まる $\overleftarrow{\delta}(i)$ から求めることができ，*DPWCFT* の最内ループによる繰り返し計算が不要であることを示唆している．

3.3 *cost*($j-1$) の改良

式 (4) を用いることにより，$i < j$ なる全ての i について $C(i, j-1)$ を求めるための関数 *cost*($j-1$) は，以下の等価な操作に置き換えることができる．まず $H(j) = \{h_1, \ldots, h_{m_j}\}$ を，j に先行する SBS の先頭の分岐番号の中で，以下を満たす k_l が存在するような $h_l = \overleftarrow{h}(k_l)$ の集合とし，$h_1 < \cdots < h_{m_j}$ とする．

- $k_l = min\{k \mid k > j,\ \overleftarrow{h}(k) = h_l\}$
- $\overleftarrow{t}(k_l) \le j$，かつ $j < k < k_l$ なる任意の k について $\overleftarrow{h}(k_l) < \overleftarrow{h}(k)$

次に $h_l \in H(j)$ について，k_l を上記の条件を満たすものとし，かつ $k_0 = \min(\{k > j \mid \overleftarrow{h}(k) = 0\} \cup \{N+1\})$ とした上で，以下の**差分コスト**を定義する．

$$\vec{\Delta}(h_l, j) = \sum_{k=j+1}^{k_{l-1}} \vec{\delta}(k) \qquad (6)$$

最後に，$i \le j$ なる各々の i について，改良版の *cost*() が計算する**ベースコスト** $\tilde{C}(i, j)$ を以下のように定義する．

$$\tilde{C}(i,j)=\begin{cases} C(i,j)+\overrightarrow{\Delta}(h_1,j) & i<h_1 \\ C(i,j)+\overrightarrow{\Delta}(h_l,j) & h_{l-1}<i\le h_l \\ C(i,j) & h_{m_j}\le i \end{cases} \quad (7)$$

なお式 (6) より $\overrightarrow{\Delta}(h_1,j)=\sum_{k=j+1}^{N}\overrightarrow{\delta}(k)$ であり，また (4) を繰り返し適用することで全ての $i<h_1$ について $C(i,j)=C(i,N)-\overrightarrow{\Delta}(h_1,j)$ が得られるので，全ての $i<h_1$ について $\tilde{C}(i,j)=C(i,N)$ である．またある $i<j$ について，$\tilde{C}(i,j)$ と $\overrightarrow{\Delta}(h_l,j)$ から $C(i,j)$ を求めるには，$H(j)$ を表現する配列に対する二分探索を行って，i に対して式 (7) が定める h_l を求めればよい．

以上に基づき，$H(j-1)$，$\overrightarrow{\Delta}(h_l,j-1)$ および $\tilde{C}(i,j-1)$ を，これらの j に関する値から求める手続を示す．この手続では $i=j$ から $i=\overleftarrow{h}(j)$ まで，降順に b_i を走査する．$i=h_{m_j}\in H(j)$ なる i が見つかるまでは，$\tilde{C}(i,j-1)$ を下式により計算する．

$$\begin{aligned}\delta(i,j)&=m_{e(j)}(i,j)-m_{e(j)}(i,j-1)\\ \tilde{C}(i,j-1)&=\tilde{C}(i,j)-\delta(i,j)\\ &=C(i,j)-\delta(i,j)=C(i,j-1)\end{aligned}$$

次に $i=h_{m_j}$ に達すると，$H(j)$ から h_{m_j} を除去し，$\tilde{C}(i,j-1)$ の計算式を

$$\begin{aligned}\tilde{C}(i,j-1)&=\tilde{C}(i,j)-\overrightarrow{\Delta}(h_{m_j},j)-\delta(i,j)\\ &=C(i,j)-\delta(i,j)=C(i,j-1)\end{aligned}$$

に置き換え，これを $i=h_l$ なる i が見つかるたびに繰り返す．また $i=\overleftarrow{h}(j)$ に達する以前に必ず $i=\overleftarrow{t}(j)$ となるので，式 (4) により $\overrightarrow{\delta}(j)=\delta(\overleftarrow{t}(j),j)$ とする．最後に $i=\overleftarrow{h}(j)$ に達すると，$H(j-1)$ を $H(j)$ の残存要素とするが，$\overleftarrow{h}(j)\notin H(j)$ であればこれを $H(j-1)$ の末尾要素として付加する．また生成された $H(j-1)$ の各要素 h_l に対して，$\overrightarrow{\Delta}(h_l,j-1)=\overrightarrow{\Delta}(h_l,j)+\overrightarrow{\delta}(j)$ とする☆．この結果，全ての $i<\overleftarrow{h}(j)$ について以下が満たされる．

$$\begin{aligned}\tilde{C}(i,j-1)=\tilde{C}(i,j)&=C(i,j)+\overrightarrow{\Delta}(h_l,j)\\ &=C(i,j-1)+(\overrightarrow{\delta}(j)+\overrightarrow{\Delta}(h_l,j))\ (*)\\ &=C(i,j-1)+\overrightarrow{\Delta}(h_l,j-1)\end{aligned}$$

なお (*) の式は式 (4) から導かれる．さて上式で重要なことは，$i<\overleftarrow{h}(j)$ なる全ての i について，$\tilde{C}(i,j-1)$ を個々に求める計算が不要であったことである．したがって，$\tilde{C}(i,j-1)$ と $\overrightarrow{\Delta}(h_l,j-1)$ から $C(i,j-1)$ を求めるために $H(j-1)$ の二分探索のための $O(\log|H(j-1)|)$ の手間をかけることにすれば，$cost(j-1)$ と等価な操作を行う手続 $\widetilde{cost}(j-1)$ を $O((j-\overleftarrow{h}(j))+|H(j-1)|)$ の手間で実現することができる．

☆ $\overleftarrow{h}(j)\notin H(j)$ であれば，$\overrightarrow{\Delta}(\overleftarrow{h}(j),j)$ は $H(j)$ から最後に除去された要素のものとし，何も除去されなければ 0 とみなす．

3.4 *DPWCFT* の改良

改良の次の対象は *DPWCFT* の最内ループであり，これは下式を $0\le i\le j$ について計算することで，最終的に $\Gamma_i(i,f)=\Gamma(i,f)$ を外側の j に関するループの終了時に得るものである．

$$\begin{aligned}\Gamma_j(i,f)&=\max(C(i,j)+\Gamma(j,f-1),\ \Gamma_{j+1}(i,f))\\ &=\max_{j\le k\le N}\{C(i,k)+\Gamma(k,f-1)\}\end{aligned} \quad (9)$$

この式 (9) と，3.2 節の式 (5) を組み合わせると，以下の $j\ge\overrightarrow{t}(i)$ なる任意の j について $\Gamma_j(i-1,f)$ を求める式が得られる．

$$\begin{aligned}&\Gamma_j(i-1,f)\\ &\quad=\max_{j\le k\le N}\{C(i-1,k)+\Gamma(k,f-1)\}\\ &\quad=\max_{j\le k\le N}\{C(i,k)+\overrightarrow{\delta}(i)+\Gamma(k,f-1)\}\\ &\quad=\overrightarrow{\delta}(i)+\max_{j\le k\le N}\{C(i,k)+\Gamma(k,f-1)\}\\ &\quad=C(i-1,j)-C(i,j)+\Gamma_j(i,f)\end{aligned} \quad (10)$$

この式により，$\Gamma_j(i,f)$ が既知であれば，$j\ge\overrightarrow{t}(i)$ については $\Gamma_j(i-1,f)$ の計算行う必要がなく，$j<\overrightarrow{t}(i)$ に対してのみ $\Gamma_j(i,f)$ を初期値として計算すればよいことがわかる．

この計算対象を限定する最適化のためには，*DPWCFT* のループ構造に 2 つの修正を加える必要がある．まず $\Gamma_j(i-1,f)$ の計算対象となる分岐番号の集合 $A(j)=\{i\,|\,i\le j<\overrightarrow{t}(i)\}$ は「不連続」となり，$i\in A(j)$ であるが $i'\in A(j)$ ではないような $i<i'\le j$ が存在することとなる．したがって $A(j)$ を $A(e,j)=\{i\,|\,e(i)=e,\ i\le j<\overrightarrow{t}(i)\}$ の全ての e に関する和集合として管理する必要がある．しかしこの管理は容易であり，特定のカウンタを共有する分岐をリンクする，すなわち b_i から $b_{p(i)}$ へのリンクをあらかじめ設定しておくことで，簡単に実現できる．

次に，b_j が SBS の末尾である場合，$A(j-1)$ を形成するために $A(j)$ に $\{i\,|\,\overrightarrow{t}(i)=j\}$ を加えるだけでなく，式 (10) にしたがって初期値である $\Gamma_j(i-1,f)$ を求める必要がある．ここで $i+1\in A(j)$ であれば $\Gamma_j(i,f)$ は計算されているので容易であるが，$i+1\notin A(j)$ であるときに注意が必要である．このような場合には，(10) を繰り返し適用すればよく，$i+1\notin A(j)$ であれば $\overrightarrow{t}(i+1)>j$ であるので必ず $A(j)$ の要素が見つかる．一般には，あらかじめ全分岐命令を操作して個々

の b_i について $a(i) = \min\{k \mid i < k,\ \overrightarrow{t}(i) \leq \overrightarrow{t}(k)\}$ を求めておけば，下式によって $\Gamma_j(i-1,f)$ を求めることができる．

$$\Gamma_j(i-1,f) = C(i-1,j) - C(a(i)-1,f) + \Gamma_j(a(i)-1,f) \quad (11)$$

以上に基づき *DPWCFT* の改良版 *OptWCFT* は，概略以下のように得られる．

Algorithm OptWCFT :

```
Γ[0...N][0] ← cost(N);  find a(i) for all i;
for f = 1 to F do begin
  Γ[0...N][f] = 0;  C̃[0...N][f] = cost(N);
  H ← ∅;  A ← A(N);
  for j = N downto 1 do begin
    for ∀i ∈ A do
      Γ[i-1][f] ← max(C(i-1,j) + Γ[i-1][f-1],
                      Γ[i-1][f]);
      A ← A - {j};
    if b_j is an SBS tail then begin
      A' ← {i | t→(i) = j};
      for ∀i ∈ A' do
        Γ[i-1][f] ← C(i-1,j) - C(a(i)-1,j) +
                    Γ[a(i)-1][f];
      A ← A ∪ A';
    end
    (H, C̃[]) ← cost~(j-1);
  end
end
```

なお A の初期値である $A(N)$ は，全てのカウンタに関する最後の SBS に後続する全分岐番号の集合である．また $C(i,j)$ は，式 (7) に基づき $\tilde{C}[i] = \tilde{C}(i,j)$ と $\overrightarrow{\Delta}(h,j)$ から求められ，h は $H = H(j)$ を i をキーとした二分探索により求められる．

3.5 *OptWCFT* の時間計算量

OptWCFT の時間計算量を見積もるために，まずいくつかの定義を行う．まず B を解析対象のワークロードが参照したカウンタの総数とする．明らかに $B \leq |P|$ であり，また仮に $|P|$ が無限大であっても（理想的分岐予測器）B はプログラム中の「静的」な分岐命令の数で抑えられる．次に SBS の先頭分岐 b_j に対して $\sigma(j) = \{i \mid \overrightarrow{h}(i) = j\}$ とし，また L を $|\sigma(j)|$ の平均値とする．さらに λ を SBS に含まれる分岐命令数の平均値とする．

まず $\widetilde{cost}(j-1)$ の手間は $O((j - \overleftarrow{t}(j)) + |H(j-1)|)$ であり，$j - \overleftarrow{t}(j)$ の平均値は $B(L+\lambda)/2$，また $|H(j-1)| \leq B$ である．したがって $\widetilde{cost}()$ の平均時間計算量は $O(B(L+\lambda))$ と見積もられる．次に $|A(j)|$ の平均値も $B(L+\lambda)/2$ であり，この値が *OptWCFT* の 2 つの最内ループの最初のものの繰り返し回数となる．このループでは，$C(i,j)$ の計算のために $\log|H(j)| \leq \log B$ の手間で $H(j)$ の二分探索を行うため，ループの平均計算量は $O(B \log B \cdot (L+\lambda))$ となる．したがって *OptWCFT* 全体の時間計算量は $O(NB \log B \cdot (L+\lambda)F)$ となる．

ここで各分岐命令の成立・不成立がランダムであると仮定して，L と λ を見積もる．なおこの仮定は我々のアルゴリズムにとって最悪のものではないが，この状況では分岐予測ミス率が 50 %ととなることから，実用上は十分に悲観的な仮定であると言える．この仮定に基づくと，ある分岐命令が SBS (TT(NT)*T または NN(TN)*N) の先頭である確率 p_S は，$p_S = 2(1/2^3 + 1/2^5 + 1/2^7 + \cdots)$ より 1/3 となる．したがって明らかに $L = 1/p_S = 3$ である．一方 λ の期待値は，SBS の長さが $2k+1$ である確率が $3(1/4)^k$ であることと，無限級数公式 $\sum_{k=1}^{\infty} kx^{k-1} = 1/(1-x)^2$ より，$\lambda = \sum_{k=1}^{\infty} 3(2k+1)/4^k = 11/3$ と求められる．したがって $L+\lambda$ は，分岐の成立・不成立がランダムであると仮定しても小さな定数 20/3 であり，後述のように実用的なプログラムではより小さな値となる．

最後に B は，N と無関係でかつ N よりも十分小さな定数であるので，*OptWCFT* の時間計算量は $O(NF)$ であると結論できる．

4. 実　　験

前章で述べた 2 つのアルゴリズム *DPWCFT* と *OptWCFT* を実装し，これらの性能を SPEC CPU95 ベンチマークと $|P| = 2,048$ の bimodal 分岐予測器を用いて評価した．プログラムは C 言語で記述し，gcc 3.2.2 で-O2 オプションを付してコンパイルし，2.0 GHz の Opeteron-246 と 2 GB のメモリからなる Linux PC で実行した．ベンチマークの統計量は**表 1** に示すとおりであり，表から L と λ の値が 3.5 節での見積もりよりも小さいことがわかる．

図 1 は，2 つのアルゴリズムの計算時間を各ベンチマークごとに示したものであり，最初の 100 万分岐命令を $F = 2$ で解析した結果である．グラフ中の破線は *DPWCFT* の計算時間であり，ベンチマークとの対応は 100 万分岐の解析時間の降順で影付きの長方形中に記載している．グラフから明らかなように，*DPWCFT* の計算時間は N^2 に比例しており，またベンチマークによらずほぼ一定の値となる．たとえば 100 万分岐の解析時間は 8,310 秒から 10,737 秒（約

表 1　ベンチマークの統計量

benchmark	$N(\times 10^3)$	B	L	λ
099.go	62,098	1,902	1.65	3.36
124.m88ksim	6,496	936	1.11	3.11
126.gcc	198,388	2,048	1.28	3.16
129.compress	3,889	427	1.39	3.25
130.li	24,233	587	1.26	3.16
132.ijpeg	97,793	1,103	1.18	3.10
134.perl	328,327	1,327	1.16	3.12
147.vortex	299,712	1,979	1.04	3.03
101.tomcatv	361,766	835	1.04	3.03
102.swim	28,859	790	1.06	3.05
103.su2cor	792,679	1,032	1.08	3.03
104.hydro2d	739,074	1,117	1.01	3.00
107.mgrid	205,374	812	1.11	3.10
110.applu	17,842	912	1.80	3.60
125.turb3d	734,122	1,075	1.26	3.20
141.apsi	100,362	1,169	1.11	3.07
145.fpppp	3,385	806	1.16	3.09
146.wave5	192,736	1,132	1.02	3.01

2.5 時間）の間に分布している．

一方，実線と影なし長方形の凡例に示す *OptWCFT* の計算時間は，ベンチマークの性質に強く依存している．グラフから明らかなように，ベンチマークは tomcatv, hydro2d, fpppp, および apsi からなる第 1 グループと，その他からなる第 2 グループに分類することができる．第 2 グループについては，*OptWCFT* の性能は *DPWCFT* を予想通り大きく凌駕し，100 万分岐の解析性能は最大で 173 倍，平均でも 80 倍である．また *OptWCFT* の時間計算量が $O(N)$ であるという予想に合致し，約 5,000 分岐／秒の一定速度で解析が進む．

その一方で第 1 グループの性能曲線，特に最も計算時間を要する 3 つのベンチマークの曲線は，他のものとは違って線形ではないように見え，3.5 節の解析の妥当性に疑問を投げかける結果となっている．しかしこれらは，単に 40 万分岐付近から急峻な傾きを持つようになる線形のものであると考えることもできる．この考え方の根拠は，これらのベンチマークでは 150 分岐以上という長大な SBS を持つような，非常に飽和しにくいカウンタが存在することにある．このような長い SBS の存在は，SBS の平均長で定まる時間計算量には影響しないが，実際の計算時間にとってはメモリアクセス局所性の極端な低下という悪影響を及ぼす．すなわち，長い SBS が間歇的に生じると，$\widehat{cost}()$ と *OptWCFT* の最内ループは非常に広い範囲に分布した多数の分岐命令を間歇的に走査することとなり，その結果キャッシュミスはもちろん TLB ミスさえも頻発する結果となって，分岐命令あたりの処理時間が

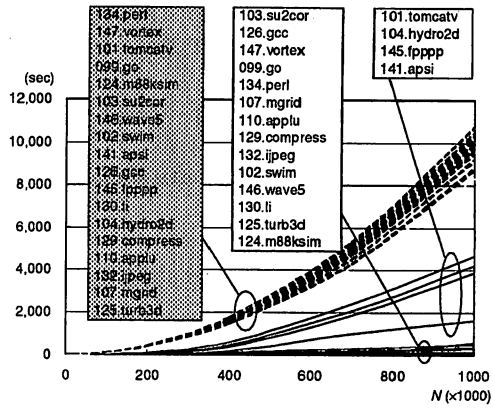

図 1　*DPWCFT* と *OptWCFT* の実行時間

長くなって時間計算量の定数項が大きくなるものと考えられる．

5. ま と め

本論文では，bimodal 分岐予測器の WCFT 解析を行う 2 つのアルゴリズムを提案した．まずこの解析が動的計画法によって実現できることを定式化し，それに基づいて $O(N^2F)$ のアルゴリズム *DPWCFT* を設計した．続いて SBS の概念を導入し，それを利用することで時間計算量を $O(NF)$ に改善したアルゴリズム *OptWCFT* が得られることを明らかにした．また SPEC CPU95 ベンチマークによる評価の結果，*OptWCFT* の性能が *DPWCFT* を大きく凌駕し，提案した最適化が重要であることが明らかになった．

謝辞　本研究の一部は文部科学省科学研究費補助金（基盤研究 (B)，研究課題番号 17300015,「高度情報機器開発のための高性能並列シミュレーションシステム」）による．

参 考 文 献

1) 宮本寛史，飯山真一，冨山宏之，高田広章，中島浩：キャッシュフラッシュの最悪タイミングの効率的な探索手法，情報処理学会論文誌コンピューティングシステム，Vol.46, No.SIG 16 (ACS12), pp.85–94 (2005).
2) Nakashima, H.: WCFT Algorithms for Branch Predictor and Proofs of Their Correctness, http://www.para.tutics.tut.ac.jp/TR/bpred-alg.pdf (2006).
3) Smith, J. E.: A study of Branch Prediction Strategies, *ISCA'81*, pp.135–148 (1981).