社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

キャッシュヒット率の向上のための基本ブロックのアドレスオフセットの探索

後藤　潤哉†　　石浦菜岐佐†

† 関西学院大学 理工学部　〒 669–1337 兵庫県三田市学園 2-1

あらまし　本稿では, プログラムの基本ブロックの前にキャッシュブロックサイズよりも細かい単位でオフセットを挿入することにより, 命令メモリのキャッシュミスを削減する手法を提案する. 本手法では, キャッシュシミュレーションに基づいて, キャッシュミス数を最小化するオフセットの組み合わせを求める. 可能なオフセットの組み合わせを全探索すると, オフセット挿入箇所の指数に比例する計算時間が必要になるため, 本稿ではシミュレーテッドアニーリングにより解の探索を行う. 1 レベルのダイレクトマッピングキャッシュを想定し, 7 つのベンチマークに対して, 実験を行ったところ, 30 箇所のオフセット挿入によって平均約 10%のキャッシュミスを削減することができた.

キーワード　キャッシュメモリ,　シミュレーテッドアニーリング,　オフセット, キャッシュミス率, 基本ブロック

Exploration of Address Offsets of Basic Blocks for Cache Hit Ratio Improvement

Junya GOTO† and Nagisa ISHIURA†

† Kwansei Gakuin University,　Gakuen 2–1, Sanda, Hyogo, 669–1337, Japan

Abstract　This article proposes a method of reducing cache misses on an instruction memory by inserting offsets before basic blocks of a given program. The addresses of the basic blocks are adjusted by a unit smaller than the cache block size. A combination of the offsets that minimizes cache miss counts, which are computed by cache simulation, is searched. Since exhaustive search would require time exponential to the number of the offsets, the solution is searched by simulated annealing. An experiment on 7 benchmarks, assuming a single-level direct-mapping instruction cache, resulted in about 10% reduction in the cache miss count on average.

Key words　cache memory, simulated annealing, offset, cache miss rate, basic block

1. はじめに

キャッシュメモリ (以下, キャッシュと略する) は, CPU とメインメモリの速度差を吸収するための高速で小容量のメモリである. プログラムの性能はキャッシュのヒット率に大きく左右されるが, キャッシュのヒット率はプログラムやデータを配置するアドレスや, キャッシュの構成等の様々な要因で変化する. このため, プログラムおよびキャッシュ構成を調整してキャッシュミスを削減する研究が多く行われている.

文献 [1] では, 与えられたプログラムを高速に実行するためのキャッシュ構成を求めている. プログラムのトレースに対するシミュレーションに基づいて, キャッシュミスを最小化するキャッシュサイズ, ブロックサイズ, 連想度の組み合わせを求める. このアプローチは, プログラムに合わせてキャッシュ側を最適化するものである.

これに対し, 与えられたキャッシュ構成に対してプログラム側を調整することによって, キャッシュミスの削減を図る研究も行われている. 文献 [2] では, プログラムとプロファイル情報を入力として, キャッシュミスが最小となるように基本ブロックの配置アドレスを決定する. 基本ブロック間の遷移確率に基づいて, 1 つ以上の基本ブロックを 「トレース」 としてまとめ, キャッシュ上での衝突を最小にするような各トレースの配置アドレスを整数線形計画法で求めている. トレースは, キャッシュブロックサイズの境界に整列することを想定しており, オフセットまで考慮した基本ブロックの配置は行っていない. また, 文献 [3] では, 最悪実行時間 (WCET) を考慮して基本ブロックの配置アドレスを決定することにより, キャッシュのコンフリクトミスの削減を図っている. この手法では, 静的コード解析に基づいて衝突グラフを作成し, 基本ブロックまたは関数の最適な配置アドレスを求める. この手法においても, 基本ブロックはキャッシュブロックサイズに整列することを想定している.

これに対し, 文献 [4] では, 基本ブロックのアドレスをキャッ

シュブロックサイズよりも細かい単位で調整することによって, キャッシュミスを削減することを試みている. 複数の基本ブロックの直前にオフセットを挿入することによってアドレスの調整を行い, キャッシュミスを最小化するオフセットの組み合わせをキャッシュシミュレーションによる全探索で求める. シミュレーションによって, 与えられたトレースに対する厳密なキャッシュミス数を求めることができるが, 探索にはオフセット挿入箇所の指数に比例する計算量が必要となる. このため, この文献では, 二分決定グラフ (BDD) [5] を用いてこのシミュレーションを高速化する手法を提案しているが, オフセットの挿入箇所は 10 箇所程度に限られる.

そこで本稿では, シミュレーテッドアニーリングを用いて文献 [4] の探索に要する時間を削減し, オフセットの挿入箇所を増やす方法を提案する. 複数の基本ブロックの前に挿入するオフセット量の組み合わせをシミュレーテッドアニーリングの状態とし, キャッシュシミュレーションにより求めるキャッシュミス数を評価関数として最小化を行う. 7 つのベンチマークプログラムに対して実験を行った結果, 命令メモリの 30 箇所にオフセットを挿入することにより, キャッシュミスを平均約 10.36%削減することができた.

以下, 本稿では, 2 章でキャッシュメモリについて要点を整理した後, 3 章で提案手法について述べる. 4 章で実験結果を示し, 5 章でまとめと今後の課題を述べる.

2. キャッシュメモリ

本稿では, ダイレクトマッピング (direct mapping) のキャッシュを扱う. また, キャッシュのレベル数は 1 とする.

メモリアクセスに用いられるアドレスは, 図 1 に示すように, 上位から tag, index (c ビット), offset (b ビット) の 3 つのフィールドに分けられる. キャッシュは, サイズ 2^b バイトのブロック (ライン) 単位でメインメモリ上のデータのコピーを管理する. キャッシュには 2^c 個のブロックが収容でき, キャッシュの容量は 2^{b+c} バイトである. キャッシュの i 番目の位置には, index フィールドの値が i であるブロックのデータが, ブロックを識別するための tag フィールドの値とともに格納される. index が i のアドレスにアクセスがあった時, そのアドレスの tag の値がキャッシュに格納されている tag と等しければキャッシュヒット, そうでなければキャッシュミスが発生する. キャッシュミスが発生した場合には, キャッシュの i 番目の エントリを新しいものに更新する.

キャッシュミスは, キャッシュ容量の不足に起因する容量性ミス (capacity miss), index が等しいアドレスのアクセス衝突に起因する (conflict miss), キャッシュブロックを最初にアクセスする時に発生する初期参照ミス (compulsory miss) に分類されるが, 本稿では特にこれを区別しない.

3. 基本ブロックのオフセット探索

3.1 オフセットによる基本ブロックのアドレス調整

本稿では命令メモリを対象に, 基本ブロックの開始アドレスをオフセットの挿入により調整する. 基本ブロックの開始アドレスは, ブロックサイズの倍数に限定しない. ただし, 基本ブロックの順序の入れ替えは行わないものとする.

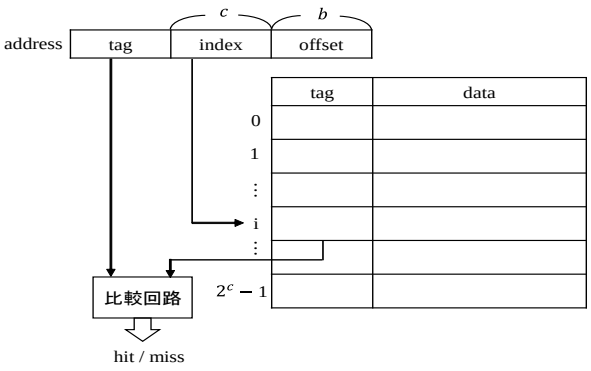


図 1　キャッシュメモリの構成

オフセットは基本ブロックの直前に挿入するが, 連続して実行される可能性がある基本ブロックの間には挿入しないものとする. 例えば, 図 2 (a) では, 条件分岐で終了する基本ブロック BB1 の次に BB2 が続いているが, BB1 と BB2 は連続して実行される可能性があるため, この間にオフセットは挿入しない. (b) の BB4 は無条件分岐で終了しているため, その直後にはオフセットを挿入しても良い. BB6 は無条件分岐の分岐先であるが, 直前の BB5 と連続して実行される可能性があるため, BB6 の前にオフセットを挿入することはできない.

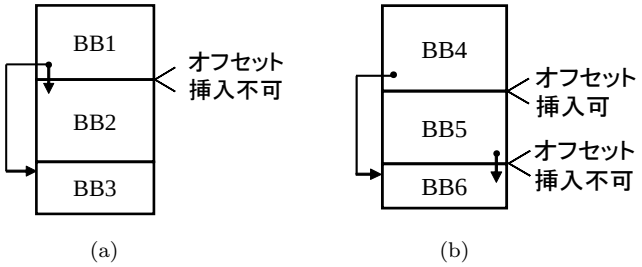


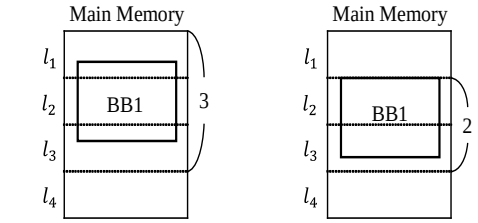
図 2　オフセット挿入箇所

基本ブロックが 1 つの場合には, 図 3 (a) のように, 基本ブロックの開始アドレスをブロックサイズの境界に整列すれば, 必ず使用ブロック数が最小になる. しかし, 複数の基本ブロックがある場合には, (b) のように, 基本ブロックをブロックサイズ境界に整列しない方が使用ブロック数が少なくなることがある. また, (c) のように, 連続する基本ブロック BB1, BB2 のうち後続 BB2 が高頻度で実行されるような場合には, BB1 をブロックサイズ境界からオフセットさせることにより BB2 の使用ブロック数を減らすことができる.

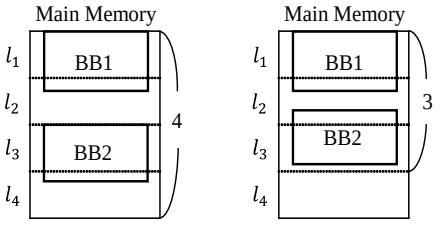
本稿では, このような考察に基づき, 基本ブロックをブロックサイズ境界に整列させる以外のアドレス調整を探索する.

3.2 キャッシュミス数を最小化するオフセットの探索

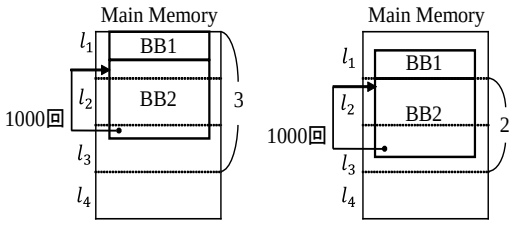
本稿では, 機械語のプログラムおよび命令メモリのトレース (実行された命令アドレスの系列) が与えられた時に, キャッシュミス数が最小となるオフセットの組み合わせを求める. ブロッ



(a) 基本ブロックが 1 つの場合



(b) 基本ブロックが複数の場合



(c) 実行回数の多い基本ブロックが後続にある場合

図 3　基本ブロックのアドレス調整

クサイズやキャッシュサイズは固定されているものとする.

オフセットは機械語プログラムの m 箇所 (前節で述べたオフセットが挿入可能な基本ブロックの先頭) に挿入するものとし, そのアドレスを, $s = (s_1, s_2, \cdots, s_m)$ (ただし, $s_1 < s_2 < \cdots < s_m$) とする. 各オフセットは, u バイト単位で 0 から始まる 2^k 段階の値を取るとする. 即ち, オフセットの値は $0, u, 2u, \cdots, (2^k-1)u$ のいずれかである. $L_k = \{0, 1, 2, \cdots, (2^k-1)\}$ とする. 以下では挿入するオフセットの段階 (単位数) をオフセットレベルと呼び, アドレス s_i に挿入するオフセットレベルを $v_i \in L_k$ とする. この時, アドレス s_i に挿入するオフセットは $v_i \cdot u$ バイトとなる. アドレス $s = (s_1, s_2, \cdots, s_m)$ に挿入するオフセットレベルが $v = (v_1, v_2, \cdots, v_m)$ の時, アドレス a は次のように定義されるアドレス $t_{s,v}(a)$ にマップされる.

$$t_{s,v}(a) = \begin{cases} a & (a < s_1) \\ a + \sum_{j=1}^{i} v_j \cdot u & (s_i \leqq a < s_{i+1}) \\ a + \sum_{j=1}^{m} v_j \cdot u & (s_m \leqq a) \end{cases}$$

s へのオフセットレベル v の挿入により, トレース $a = a_1 a_2 \cdots a_n$ は, $t_{s,v}(a) = t_{s,v}(a_1) t_{s,v}(a_2) \cdots t_{s,v}(a_n)$ にマップされる. トレース $t_{s,v}(a)$ に対するキャッシュミス数を $m_{s,a}(v)$ とすると, 本稿で扱うキャッシュミス数最小化問題は, a と s が与えられた時に, $m_{s,a}(v)$ を最小にする v を求める問題である.

3.3 探索手法

本稿では, この最小化問題の解を, メタヒューリスティクスの 1 つであるシミュレーテッドアニーリング法 (simulated annealing) [6] により求める.

オフセットレベル $v = (v_1, v_2, \cdots, v_m)$ をそのまま探索の状態とし, $m_{s,a}(v)$ を目的関数として最小化する. v が与えられた時の $m_{s,a}(v)$ は, キャッシュシミュレーションにより求める.

v に対し, i 番目のオフセットレベルのみが異なっており, その差が d 以下であるようなオフセットレベルの集合を $N_{v,i,d}$ とする.

$$N_{v,i,d} = \{u \in L_k^m \mid v_j = u_j \ (for\ j \neq i), u_i \neq v_i, \mid u_i - v_i \mid \leqq d\}$$

シミュレーテッドアニーリングの総冷却回数が C, ある時点までの冷却回数が c の時, $r = (C-c)/C$ をその時点での冷却回数の残率と呼ぶことにする. 冷却回数の残率 r におけるオフセットレベルの許容変動幅 Δ_r を次のように定義する.

$$\Delta_r = (1-r) + \frac{2^k}{2r}$$

この時, 本手法では, 冷却回数の残率 r における状態 v の近傍 $N_{v,r}$ を次のように定義する.

$$N_{v,r} = \bigcup_{i=1}^{m} N_{v,i,\Delta_r}$$

シミュレーテッドアニーリングの初期状態は L_k^m からランダムに選び, v の次状態は, $N_{v,r}$ からランダムに選ぶ.

4. 実験結果

提案手法に基づくオフセット探索プログラムを C 言語 で実装した. 本プログラムは Windows Cygwin, Ubuntu Linux 等の環境で動作する.

実験には 7 つのベンチマークを ARM 用にコンパイルしたものを用いた. 各プログラムのトレース長と基本ブロックは数は表 1 に示すとおりである.

表 1　ベンチマークプログラムの諸元

プログラム	トレース長	基本ブロック数
mips	29,853	224
mpeg2	35,661	256
gsm	52,659	287
dfmul	102,079	431
dfdiv	179,681	497
adpcm	189,242	283
dfadd	251,911	478

表 2 は, ベンチマークのうちトレース長の短い 3 つについて, オフセット挿入箇所数 m を 4 に限定して実験を行った結果である. 各オフセットの値は 4 バイト単位の 32 段階 (最小 0 バイト, 最大 124 バイト) で探索した. キャッシュブロックサイズは 32, キャッシュサイズは 64 または 256 である. 「全探索」 は, 可能なオフセットの全組み合わせに対してシミュレーションを

表 2　キャッシュミス数の比較
(オフセット挿入箇所数 m=4)

プログラム	キャッシュサイズ	キャッシュミス数	
		全探索	SA
mips	256	24,795	24,795
mpeg2	64	30,883	30,883
gsm	256	46,360	46,360

オフセット:　4 バイト単位, 32 段階
ブロックサイズ:　32

表 3　基本ブロックのキャッシュブロック境界からのオフセット

プログラム	オフセット [バイト]
mips	$(16, 20, 0, 8) \times 2$
mpeg2	$(16, 24, 20, 8) \times 144$
gsm	$(0, 4, 0, 4) \times 1$, $(0, 4, 24, 4) \times 3$, $(0, 12, 0, 4) \times 1$, $(0, 12, 24, 4) \times 3$

行う方法である.「SA」は, 本稿のシミュレーテッドアニーリングであり, 冷却率は 0.95, 冷却回数は 90 回, 反復回数は 1000 回に設定した. この規模の問題では, シミュレーテッドアニーリングで厳密な最小解が得られている.

表 3 は, 表 2 の実験の全探索において, キャッシュミス数を最小にする全てのオフセットの組み合わせについて, 基本ブロックのキャッシュブロック境界からのオフセットを列挙したものである. 例えば, mips の「 (16, 20, 0, 8) 」は, キャッシュミス数が最小になった時の 4 つの基本ブロックのキャッシュブロック境界からのオフセットが, それぞれ 16, 20, 0, 8 バイトであったことを表す.「×2」は, キャッシュミス数を最小にするオフセットの組み合わせは全部で 2 通りあったが, キャッシュブロック境界からのオフセットはいずれも等しかったことを表す. mpeg2 では 144 通りの組み合わせについて, キャッシュブロック境界からのオフセットは全て等しかった. gsm は 8 通りの組み合わせが 4 パターンに分類された. 4 つの基本ブロックが全てキャッシュブロック境界に整列されている場合には, このオフセットは (0, 0, 0, 0) となるが, 表 3 にはこれが現れていない. これは, ブロックサイズよりも細かい単位で基本ブロックのアドレスを調整することにより, キャッシュミスを削減できることを意味する.

オフセット挿入箇所数 m を 20〜40 に増やして実験を行った結果を表 4 に示す. 表 2 の実験と同様, 各オフセットの値は 4 バイト単位の 32 段階とし, キャッシュブロックサイズは 32 とした. キャッシュサイズは 表 2 に示した通りである. 挿入箇所は, 実行回数が最も多い m 個の基本ブロックの前とした. シミュレーテッドアニーリングの冷却率は 0.97, 冷却回数は 152 回, 反復回数は 1000 回とした.

$m = 0$ はオフセットを全く挿入しない場合であり, オフセット挿入によりキャッシュミス率が削減できていることがわかる. $m = 30$ の時に最も効果が大きく, キャッシュミス率が平均で 2.44% 削減できている. これは, キャッシュミスが 10.36% 削減できていることに相当する. m を増やしてもヒット率が減少しないプログラムが見られるが, これは実行回数が多い基本ブロックの数が m より小さくなり, 削減効果がほとんどない空間を探索しているためと考えられる.

表 5 は探索に要した時間である. CPU は Core i7, メモリは 7.7GB, OS は Ubuntu 14.04 である. キャッシュミス数をシミュレーションで求めているため, 探索時間はトレース長に比例して増加する. 現在, アドレスのマッピングを線形探索で求めているため, m に比例する計算時間が必要になっているが, この点には改善の余地がある.

表 4　提案手法のキャッシュミス率

プログラム	キャッシュサイズ	キャッシュミス率			
		m=0	m=20	m=30	m=40
mips	256	17.07%	9.81%	8.75%	8.17%
mpeg2	64	13.45%	13.36%	13.36%	13.35%
gsm	256	12.05%	8.15%	7.79%	7.78%
dfmul	4096	6.93%	5.39%	5.24%	5.69%
dfdiv	1024	9.48%	8.84%	8.62%	8.28%
adpcm	128	10.41%	9.98%	9.98%	9.98%
dfadd	1024	11.55%	9.89%	10.15%	10.75%
増減 (平均)			−2.22%	−2.44%	−2.42%
キャッシュミス削減率 (平均)			9.72%	10.36%	9.82%

m: オフセット挿入箇所数
オフセット:　4 バイト単位, 32 段階
ブロックサイズ:　32

表 5　探索時間

プログラム	探索時間 [秒]		
	m=20	m=30	m=40
mips	331.60	511.24	740.23
mpeg2	436.53	626.65	766.99
gsm	581.85	849.07	1237.85
dfmul	1133.50	1676.44	3286.88
dfdiv	2011.96	3002.86	4086.46
adpcm	1966.37	3123.24	3968.00
dfadd	2983.11	4252.70	5966.49

CPU: Core i7 1.80GHz, メモリ: 7.7GB

5. むすび

本稿では, プログラムの基本ブロックの前にキャッシュブロックサイズよりも細かい単位でオフセットを挿入することにより, 命令メモリのキャッシュミスを削減する手法を提案した. 7 つのベンチマークに対して, 実験を行った結果, 平均約 10.36%のキャッシュミスを削減することができた.

キャッシュミスは個々の要因だけでなく, 様々な要因の組み合わせでも大きく変化するため, これらを考慮した最適化について検討することが今後の課題として挙げられる.

謝辞　本研究を進めるにあたり, 多くのご助言やご協力を頂いた関西学院大学理工学部 石浦研究室の諸氏に感謝します.

文　　献

[1] Nobuaki Tojo, Nozomu Togawa, Masao Yanagisawa and Tatsuo Ohtsuki: “Exact and Fast L1 Cache Simulation for Embedded Systems,” in Proc. Asia and South Pacific Design Automation Conference (ASP-DAC 2009), pp. 817–822 (Jan. 2009).

[2] Hiroyuki Tomiyama and Hiroto Yasuura: “Code Placement Techniques for Cache Miss Rate Reduction,” ACM Transactions on Design Automation of Electronic Systems (TODAES) Vol. 2, No. 4, pp. 410–429 (Oct. 1997).

[3] Heiko Falk and Helena Kotthaus: “WCET-Driven Cache-Aware Code Positioning,” in Proc. 14th International Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES ’11), pp. 145–154 (Oct. 2011).

[4] 林憲太郎, 平見健太, 石浦菜岐佐: “オフセット探索のためのキャッシュシミュレーションの二分決定グラフによる高速化,” 電子情報通信学会ソサイエティ大会, A-3-3 (Sept. 2015).

[5] 石浦菜岐佐: “BDD とは,” 情報処理, 第 34 巻, 第 5 号, pp. 585–592 (May 1993).

[6] 喜多一: “シミュレーテッドアニーリング,” 日本ファジィ学会誌 Vol. 9, No. 6, pp. 870–875 (Dec 1997).