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

ACCELERATING CACHE SIMULATION FOR OFFSET EXPLORATION BY BINARY DECISION DIAGRAMS

林憲太郎 Kentarou Hayashi　平見健太 * Kenta Hirami　石浦菜岐佐 Nagisa Ishiura

関西学院大学 理工学部
School of Science and Technology, Kwansei Gakuin University

1 はじめに

キャッシュメモリーの同じブロック (ライン) にアクセスが集中すると, キャッシュミスが頻発してプログラムの実行速度が低下することがある. これを防ぐ一手法として, プログラムのコードやデータの可能な部分に間隙 (オフセット) を挿入してアドレスを調整する方法が知られている. 挿入するオフセットの可能な全ての組み合わせに対してキャッシュ動作のシミュレーションを行えば, ヒット率を最大にするオフセットの組み合わせを求めることができるが, このためにはオフセットの挿入箇所の指数に比例する計算時間が必要となる. そこで本稿では, 二分決定グラフ (BDD) [1] を用いてこの計算を高速化する手法を提案する.

2 キャッシュシミュレーション

簡単のため, 以下では主記憶のアドレスは 32 ビットで表されるものとする. オフセットは機械語やデータの m 箇所に挿入するものとし, そのアドレスを $s_0, s_1, \cdots, s_{m-1}$ とする. 主記憶上の機械語やデータのアドレスの調整は u バイト単位で 2^k 段階に行う. すなわち, アドレスの変更幅 (オフセット) は最小 0 バイト, 最大 $(2^k-1)u$ バイトであるとする. メモリーアクセスのトレース (アクセスしたアドレスの系列) およびキャッシュの容量とブロックサイズが与えられた時に, キャッシュヒット数が最大になるようなオフセット値の組合せを求めるのが本稿の目的である. ただし, 本稿ではダイレクトマッピング方式のキャッシュのみを扱う.

アドレス a に対して, a と同じキャッシュブロックに含まれるアドレスの集合を B_a, a とキャッシュインデックスが等しいアドレスの集合を C_a とする. キャッシュに収容されているアドレスの集合が M のときにアドレス a にアクセスがあった場合, $a \in M$ であればキャッシュヒット, そうでなければキャッシュミスであり, アクセス後に M は $(M \cap \overline{C_a}) \cup B_a$ に更新される. 与えられたアドレス系列に対し, $M = \phi$ を初期値としてこの計算を繰り返せば, キャッシュヒット率を求めることができる.

可能な全てのオフセットの組合せについてこの計算を行えば, 最適なオフセットの組合せを求められるが, 場合の総数は 2^{mk} となり, m や k が大きくなると膨大な時間がかかる.

3 二分決定グラフによるシミュレーションの高速化

本稿ではこれを BDD により高速化する方法を提案する. アドレスの各ビットに対応する論理変数 $x = (x_{31}, x_{30}, \cdots, x_0)$ を導入し, アドレスの集合 A を $f_A(x) = 1$ iff $x \in A$ なる論理関数 f_A で表現すると, 前節の集合演算は論理演算に置き換えることができる. さらに, アドレス s_i に挿入する 2^k 段階のオフセットを論理変数 $v_i = (v_{i,0}, v_{i,1}, \cdots, v_{i,k-1})$ で符号化し, 2^{mk} 通りのシミュレーションを一度に行う. これを BDD で表現して実行することにより, 処理を高速化する.

4 実験結果

提案手法を, 北大版 BDD/ZDD パッケージ [2] を用いて C++ で実装した. 実行時間とオフセット探索によるヒット率の改善結果を表 1 に示す. 使用したトレースは ARM 用コードの命令メモリーへのアクセス系列であり, #access はトレース長である. 本手法は, 単純な繰返し法を C で実装したもの (単純法) に比べて平均で 12.1 倍高速である. シミュレーションにより求めたオフセットを挿入した場合 (+offset), 挿入しない場合 (original) に比べてヒット率を最大 0.97 % 改善できている.

表 1 実験結果

trace	#access ($\times 10^3$)	m	u	2^k	CPU time [s] 単純法	CPU time [s] 本手法	hit ratio original	hit ratio +offset
mpeg2	35.6	6	4	16	1946.2	223.4	97.58%	97.61%
mips	29.8	5	4	16	530.4	22.7	64.44%	64.87%
gsm	52.6	5	4	16	780.7	32.5	76.72%	77.69%
aes	94.8	5	4	16	1338.8	41.7	71.50%	71.50%
bf	1,013.1	5	4	16	1768.0	215.5	81.98%	81.98%
dfadd	251.9	5	4	16	791.8	233.6	73.77%	73.87%
dfdiv	179.6	5	4	16	2129.7	1783.4	73.36%	73.45%
dfmul	102.0	5	4	16	1295.3	46.5	72.78%	73.07%
adpcm	189.2	5	4	16	1797.9	47.4	82.27%	82.35%

Ubuntu 14.04 LTS, Core i5-4310U 2.00GHz
m: オフセットの挿入箇所数, u: オフセットの単位 [B]
2^k: オフセットの調節段階数
キャッシュサイズ 2^4, ブロックサイズ 2^7

5 むすび

本稿では, オフセット探索のためのシミュレーションを BDD により高速化する手法を提案した. 今後の課題としては, 本手法のセットアソシアティブ方式への拡張が挙げられる.

参考文献

[1] 石浦菜岐佐: “BDD とは,” 情報処理, vol. 34, no. 5, pp. 585–592 (May 1993).

[2] 湊真一: “北大版 BDD/ZDD パッケージの概要,” 2010 年度 ERATO 湊離散構造処理系プロジェクト講究録, pp. 10–12 (June 2011).

*現在 エヌ・ティ・ティ・コムウェア株式会社