キャッシュ量を動的に変化させるメモ化手法

長野俊輔† 前田敦司‡ 山口喜教‡
筑波大学第三学群情報学類†
筑波大学大学院システム情報工学研究科‡

1．背景

一度呼び出した関数の引数と結果を保存しておき，同じ引数で関数が呼ばれたときに保存しておいた値を利用し，再計算を防ぐメモ化(memoization)と呼ばれる手法がある．メモ化を用いることで指数時間計算量の問題を，線形の計算量に抑えることができる場合もある．計算済みの結果をメモリに記憶しておくトップダウン動的計画法はメモ化の応用例とみなす事もできる．

ある関数を自動的にメモ化することは容易であるが，その場合すべての入力に対する値をメモ化すると，空間計算量が最悪で入力と同等に増加してしまうという問題がある．

2．目的

メモ化した値は忘れてしまっても再計算が必要なだけであり，計算結果に影響がない単なるキャッシュである．プログラムによっては，実行したものすべてをメモ化しておかなくても十分高速に動作するのではないかと考えられる．

任意の関数に対して自動的にメモ化を行い，さらにメモリ量を自動的に最小化するシステムを実装するための最初のステップとして，まず，本研究ではキャッシュ量と速度の関係を調査する．また，キャッシュ量を現在よりさらに増やすことで，よりプログラムが高速に動作するかを判断し，キャッシュ量を動的に変化させる手法を提案する．

3．提案手法

関数呼び出しに局所性を仮定すると，一度呼び出された関数はメモ化され，近いうちに再度同じ引数で呼び出され，メモ化データが利用されることになる．そこで CPU に用いられているキャッシュ置換アルゴリズムを用いることで，キャッシュ量を抑えつつも効率よくメモ化データをキャッシュでき，プログラムを高速に動作させることができると考えられる．

キャッシュ量を抑えるために，実装が簡単であり，オーバヘッドの小さいキャッシュ格納方式であるダイレクトマップ方式を用いて実験を行う．ダイレクトマップ方式はある入力に対して一意に格納場所が決まるため，同一エントリに異なる入力が転送されると必ず衝突が起き，バケットに格納されているデータは直ちに追い出される．

まず予備実験として，最長共通部分文字列問題(Longest Common Subsequence)と呼ばれる 2 つの文字列の共通部分文字列を求める問題を用いた．この問題も入力文字長により指数関数的に計算時間が増加する問題である．キャッシュサイズを変化させ，関数の呼び出し回数の削減率を求めた結果を図 1 に示す．なお，ハッシュ関数はポインタをキーにとる以下の関数とした．

hash(x,y)=((x << 4) XOR y) mod CacheSize

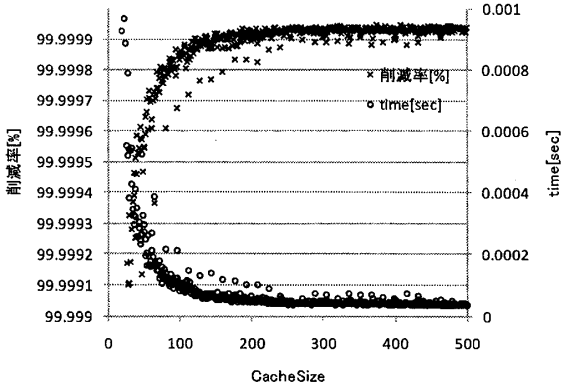

図 1 キャッシュサイズによる
LCS の性能特性の変化

キャッシュサイズを大きくすると，削減率が増加し，実行時間が減少していることわかる．キャッシュサイズと実行時間はトレードオフの関係にあり，すべてをメモ化しておかなくてもプログラムが十分高速に動作することが確認できた．

Memoization Technique with Dynamic Cache Size Adjustment
†NAGANO Shunsuke, College of Information Sciences, Third Cluster of Colleges, University of Tsukuba
‡MAEDA Atusi, YAMAGUCHI Yoshinori, University of Tsukuba Graduate School of System and Information Engineering

CPU のキャッシュにおいて，キャッシュまたは下位の記憶装置のどちらからデータを得るかによりコストは異なるが，トータルのアクセス回数は変わらない．よって，コストの小さいキャッシュへのヒット率はプログラムが高速に動作するかの重要な指標となる．しかしメモ化のためのキャッシュにおいて，キャッシュにヒットした場合，それを計算するための再帰呼び出しは不要となる．よって分母であるトータルの関数呼び出しの回数が変わってしまうため，ヒット率向上は実行速度向上を意味しない．そのため，キャッシュのヒット率を，現在のキャッシュ量で十分かを判断する指標として使うことはできない．

そこで大きさの異なる 2 つのキャッシュを用いることを提案する．便宜上，小さいサイズのキャッシュを L1，大きいサイズのキャッシュを L2 とし，2 つのキャッシュは共に同じ構造とする．L1・L2 は速度ではなく，集合の包含の階層となっている．関数を呼び出す際，

- L1 にメモ化データがあればそれを用いる
- L1 にメモ化データがなく L2 にある場合，L1 にメモ化データをコピーしそれを用いる
- L1・L2 共にメモ化データがない場合，関数を呼び出し，結果を L1・L2 の両方に格納する

とする．このようにキャッシュの階層構造を持つようにすることで，L2 のヒット数は，L1 を L2 と同じ大きさにした場合に，どの程度追加的なヒット数の増加が見込めるかという指標になる．

4．実装

L2 の大きさを L1 の 2 倍とし，キャッシュの大きさを 2 倍，4 倍…と変えたときの関数の呼び出し回数および，L1 のヒット数に対する L2 のヒット数の比（L2/L1 比）を測定した．最長共通部分文字列問題に対して 40 文字と 20 文字の入力を与えた結果を図 2 に，与えられた金額をコインの集合で両替する際に何通りの方法があるかを求める両替問題[4]に対して，総額として 10000 セントを，コインの集合として米ドル 12 種類（単位セント）を与えた結果を図 3 に示す．なお，Java 言語で実装を行い，2 引数の関数ではそれぞれの引数のオブジェクトの hashCode の排他的論理和をとったものをハッシュ関数とした．

5．考察

図 2 の結果では，L2 のヒット数の比が大きい時は，実際にキャッシュを大きくすることにより，関数呼び出しの回数が削減でき，提案手法が有効であると考えられる．しかし，図 3 の結果のように，問題によってはうまくいかない場合がある．2 引数の関数の場合，ハッシュ関数によっても結果は大きく変わってしまう．さまざまな問題について安定して結果が得られるようにすることを今後の課題とし，さらに研究を進めていく．

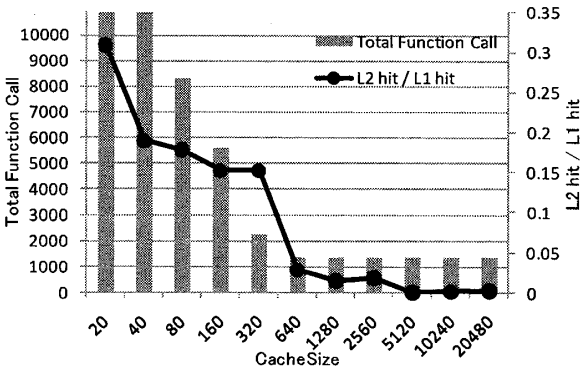

図 2 LCS の L2/L1 比の推移

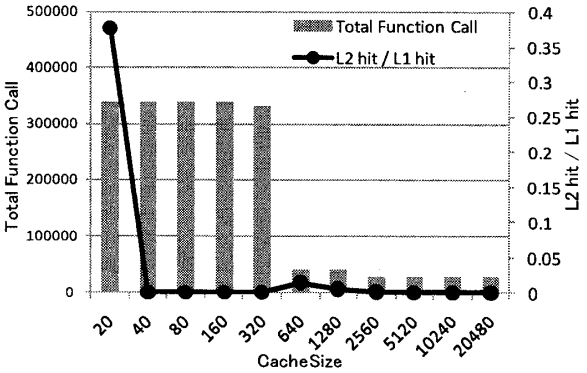

図 3 両替問題の L2/L1 比の推移

参考文献

[1] H. S. Stone, J. Turek, J. L. Wolf, “Optimal partitioning of cache memory”, IEEE Transactions on Computers, pp. 1054-1068, 1992

[2] 田中 淳裕, “キャッシュのモデル化とその応用”, Operations research as a management science research, 49(7), pp. 434-437, 2004

[3] 小川 周吾, 入江 英嗣, 平木 敬, “置換データの性質に着目した動的キャッシュパーティショニング”, 情報処理学会研究報告書, Vol. 2009-ARC-184 No. 20, 2009

[4] Harold Abelson, Gerald Jay Sussman, Structure and Interpretation of Computer Programs, The MIT Press, 1996, Chap. 1