4N-4

高速自動微分のためのプロセス制御

舟崎 智誠　久保田 光一
中央大学大学院理工学研究科*

1 はじめに

高速自動微分法は, n 変数関数の勾配を関数値計算の定数倍の手間で計算するが, 計算履歴の保存が必要なため手間に比例する記憶領域を必要とする. これは, 大規模な問題を扱うときに大きな制約となる. そこで, さまざまな記憶領域削減法が提案されている. その中の一つとして, A. Griewank らのプログラムの実行状態の保存と復元を前提とした記憶領域削減法がある [1, 2]. プログラムの実行状態を保存しておくことをプログラムの凍結とよび, 凍結した状態から再びプログラムを実行状態に戻すことをプログラムの解凍と呼ぶことにする. この方法を実装するためにはプログラムの凍結と解凍が実行できなければならないが, C 言語ではこのような機能は用意されていない. そこで本稿では, UNIX のシステムコールを用いてこの記憶領域削減法を C 言語で実装し, 計算実験を行った結果を報告する.

2 Griewank の領域削減法

前述したように, 計算履歴の保存が必要となる高速自動微分法には多大な記憶領域が必要となる. しかし, 計算履歴を保存する代わりに計算過程を再計算することで記憶領域の削減を実現することができる.

2.1 再計算による記憶領域の削減

f を 1 つの基本演算として

$$s_{i+1} \leftarrow f_i(s_i) \quad \text{for} \quad i = 0, 1, \ldots, n-2, n-1$$

という手続きで定義される関数を考える. この関数を高速自動微分する手続きを

$$\bar{s}_{i-1} \leftarrow \bar{f}_i(\bar{s}_i) \quad \text{for} \quad i = n-1, n-2, \ldots, 2, 1$$

とする. ここで, 初期状態 s_0 から目的の $\bar{s}_0$ を得るためには計算過程全ての s_i を保存しておかなければならない. しかし, 初期状態 s_0 から s_n を計算し, その値を使って $\bar{s}_{n-1}$ を計算した後で, s_0 からもう一度 $\bar{s}_{n-2}$ を計算し直せば計算履歴の保存は不要となる. このように再計算を繰り返し行うことにより, 記憶領域の削減を実現できる.

2.2 計算過程の分割

上記の計算過程を部分計算過程

$$F_j \equiv [f_{i_j}, f_{i_j+1}, \ldots, f_{i_{j+1}-1}] \quad \text{for} \quad j = 0, \ldots, p-1$$

に分割する. このとき, $s \leftarrow F_j(s)$ は計算ステップ F_j の関数値計算を表す. また, $s \leftarrow \hat{F}_j(s)$, $s \leftarrow \overline{F}_j(s)$ はそれぞれ計算ステップ F_j の計算履歴の保存をともなう関数値計算, 偏導関数値計算を表すとする.

2.3 Griewank のアルゴリズム

凍結と解凍を用いて部分計算過程の単位で再帰的な再計算行う Griewank の領域削減アルゴリズムのは図 1 ように表される [1].

```
treeverse(δ, τ, β, σ, φ){
  if (σ > β){
    δ = δ − 1
    snapshot(s)
    s ← F_j(s)   for   j = β, ..., σ − 1
  }
  while (κ =mid(δ, τ, σ, φ) < φ){
    treeverse(δ, τ, σ, κ, φ)
    τ = τ − 1
    φ = κ
  }
  s ← F̂_j(s)
  s ← F̄_j(s)
  if (α > β)    retrieve(s)
}
```

図 1 Griewank のアルゴリズム

δ, τ はそれぞれ凍結の数, 再帰の深さをカウントしている. σ, ϕ はその呼び出しがカバーする計算過程の

*Process Control Strategies for Reverse-mode Automatic Differentiation, Tomoshige FUNASAKI and Koichi KUBOTA, Graduate School of Science and Engineering, Chuo University, 1-13-27 Kasuga, Bunkyo-ku, Tokyo 112-8551, Japan.

始点と終点を表している．また，**snapshot** はプログラムの凍結を，**retrieve** はプログラムの解凍を行う．関数 **mid** は σ と ϕ の間の適当な値を返す．

3 プロセスの制御

Griewank のアルゴリズムで用いらている関数 **snapshot**, **retrieve** のような働きをする機能が直接用意されている環境は少ない．そこで，UNIX のシステムコール **fork** とセマフォを使って Griewank の領域削減アルゴリズムを実装する．

3.1 凍結，解凍の機能の実現

fork で作られた子プロセスは，**fork** を実行した親プロセスのコピーである．その子プロセスの動作を，セマフォを使って制御することにより凍結，解凍の機能を実現する．ただし，**snapshot**，**retrieve** と同じ働きをさせるには，プログラムの制御構造を変える必要がある．

3.2 制御方式

プロセス制御のやり方にはいろいろあると思われるが，今回の発表では **fork** で作られた子プロセスを 1 つのプロセスで集中的に制御する方式で行う．制御するプロセス (サーバープロセス) は，[1] の理論に基づいた計算のスケジューリングを行う．そして，そのスケジュールに従いクライアントプロセスに必要な計算を実行させる．

3.3 アルゴリズムの実現

上記のプログラムの凍結，解凍の機能を使って高速自動微分の領域削減を実現するために以下の 3 つの手順を用意する．これらの手順を 1 つの実行単位として，サーバープロセスは各プロセスにどの手順の動作をするかを指示する．

1. 凍結の複製
既にある凍結を複製する．具体的にはある凍結を解凍し，そこで **fork** を実行して新しくプロセスを作る．

2. 関数値計算と凍結
関数値計算を実行した後で，**fork** を実行して新しくプロセスを作り，凍結をする．

3. 関数値計算と偏導関数値計算
計算履歴の保存を伴う関数値計算を行い，記録した計算履歴をもとに偏導関数数値の計算を行う．その後，プロセスを消滅させる．

このような 3 つの手順を使って $\delta = 2, \tau = 3$ で高速自動微分法の領域削減アルゴリズムを実行したときの様子を図 2 に示す．

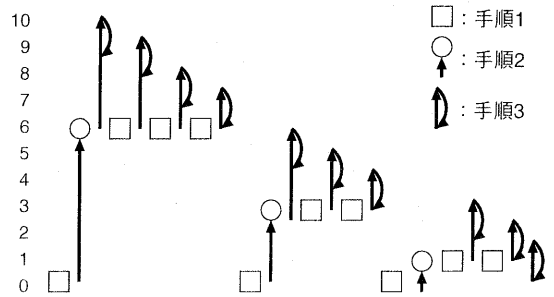

図 2 $\delta = 2, \tau = 3$ での手順

4 まとめ

UNIX のシステムコール **fork** とセマフォ [5] を用いた凍結，解凍の機能を使い Griewank のアルゴリズムを実装した．計算実験では常微分方程式の初期値問題の数値解法にこのアルゴリズムを適用し，記憶領域が削減されていることを確認した．

今回の計算実験では計算過程の部分計算過程への分割が容易に行える例を取り上げたが一般にプログラムで与えられた関数の計算過程の部分計算過程への分割は容易ではなく，コンパイラに手を入れるなどしなければならない [3, 4]．また，**fork** で複製するプロセスが多くの記憶領域を使っていると複製を作ためにシステムに大きな負担をかけてしまう．これらの事への対応が今後の課題である．

参考文献

[1] A. Griewank, "Achieving logarithmic growth of temporal and spatial complexity in reverse automatic differentiation," *Optimization Methods and Software*, 1992, Vol. 1, pp. 33–54.

[2] A. Griewank and A. Walther, "An implementation of checkpointing for the reverese or adjoint mode of comuputational differentiation," *ACM Transactions on Mathematical Software*, March 2000, Vol. 26, No. 1, pp. 19–45.

[3] Koichi KUBOTA, "A FORTRAN77 preprocessor for reverse mode automatic differentiation with recursive checkpointing," *Optimization Methods & Software*, 1998, Vol. 10, pp. 319–335.

[4] 久保田光一, 伊理正夫, "アルゴリズムの自動微分と応用," コロナ社, 東京, 1998.

[5] W. R. Stevens, 篠田陽一 訳, "UNIX ネットワークプログラミング," トッパン, 東京, 1992.