ハイパフォーマンス
コンピューティング 63－1
（1996．10．4）

Fibonacci数列などを教材に使った高性能計算の実践報告

藤野 清次　児島 彰
広島市立大学 情報科学部 情報工学科

概　要

プログラミング言語の講義や教科書では，再帰型 (recursive) のプログラミングを教える例題として，Fibonacci 数列が取り上げられることが多い．ところが，再帰型のプログラミングは，簡便性という長所はあるものの，高性能計算の立場からすると必ずしも良い教材とは言い難いときがある．そこで，本報告では，この Fibonacci 数列を始めとして，組合せの数: ${}_nC_r$ を求める問題，積和計算とコンパイラの問題，などの課題に対して，学生の数値実験結果からいくつかを選んで紹介する．

Recursive Programming in view of H.P.C.

Seiji Fujino and Akira Kojima
Hiroshima City University

Abstract

In the lecture of programming language including C, PASCAL and Fortran90, the recursive programming, i.e., Fibonacci series, factorial calculation and Combination ${}_nC_r$ are often adopted. This recursive programming is very useful and educational material for describing a recursive relation. On the other hand, it is known that recursive programming includes wasteful calculation in view of high performance computing. In this contribution, efficiency of recursive programming is discussed through the results of numerical experiments by students.

1 はじめに

プログラミング言語 (C, PASCAL, Fortran90) などでは，再帰型のプログラムを作成することができる．そのため，多くの教科書 (参考文献参照) で再帰型のプログラミングについて解説がなされている．ただ，そこでの主張は，身近なアルゴリズムの中には再帰的な関係で表されるものが多くあるので，そのような場合には，再帰型プログラミングは簡潔でわかりやすくかつ非常に便利だというものが多い．

一方，反対の側面，すなわち，再帰型であるが故に，計算効率の面からは無駄が多い (同じ関数を何度も使う) ことを指摘している文献の数は限定される．そこで，本発表では，実際に当大学の情報工学科の 2 年生と 3 年生に与えた課題の報告レポートを通して，再帰型プログラミングの効率について考察を行うことにする．

2 Fibonacci 数列

まず，情報工学科 2 年生に与えた課題を紹介する．

> [課題 1]: Fibonacci 数列
> $f_0 = 0, f_1 = 1, f_{n+2}=f_{n+1}+f_n, n \geq 0$ を求めるプログラムを (i) 再帰型，(ii) 繰返し型の 2 つ作成せよ．
> 次に，そのプログラムを使って f_{40} を求め，計算時間について 2 つの方法を比較せよ．

このとき，PASCAL で書かれた次の再帰型のプログラム (これを原案とする) を参考に示した [7]. そして，計算時間の観点から，プログラムは適宜工夫を加えるように指示を与えた．

原案のプログラム (再帰型)

```
function fibo(i:integer):integer;
begin
   if i=0 then fibo:=0
   else if i=1 then fibo:=1
   else fibo:=fibo(i-1)+fibo(i-2)
end;
```

この課題の締切は 10 日後．学生は，全員 1 年のとき C 言語を習得済み．また，計算機は実習用の WS SPARC Station SS/5 を使った．同様に，次の PASCAL 版の繰返し型のプログラム (同じく原案とする) も示した [7].

原案のプログラム (繰返し型)

```
function fibo(i:integer):integer;
 var a,b,w,n:integer;
begin
 a:=0; b:=1;n:=1;
 while n<> i do begin
 w:=b;
 b:=a+b;
 a:=w;
 n:=n+1
 end;
 fibo:=a
end;
```

この課題に対して，出来上がったプログラムを用いて，$f(40)$ を計算するのに要した再帰型のときの CPU 時間を Table 1,2 に示す．ただし，Table 2 は 10000 回の合計時間である．また，括弧の中の数字は，原案を 100 としたときの比率である．

Table 1. CPU time in seconds by recursive program for Fibonacci F(40).

再帰型 program	最適化あり	最適化なし
原案	116.8 (100.)	161.8 (100.)
改良 1	102.1 (87.4)	135.6 (83.8)
改良 2	58.9 (51.9)	85.4 (52.8)
改良 3	31.0 (26.5)	42.0 (26.0)
改良 4	0.30 (.003)	0.40 (.003)

Table 2. CPU time of 10000 times by non-recursive program for F(40).

繰返し型 program	最適化	最適化なし
原案	0.04 sec	0.11 sec
節約版	0.03 sec	0.11 sec

次に，再帰型のプログラムの改良1~4を示す．改良のポイントは，[**改良 1**] では初期値 f_0=0, f_1=1 の関係をうまく使って，if 文を 1 つ省略したものであり，[**改良 2**] は初期値の深さを 1 にしたものである．Table 1 からわかるように，[**改良 2**] の計算時間は原案の約半分になり，改善効果が大きいことがわかる．一方，[**改良 3**] は，if 文の代わりに switch 文を使って時間を節約したものであり，[**改良 4**] は，再帰型と繰返し型のハイブリッド版とも呼べる，深さ (の大きさ) を調節できる教育的プログラムとも呼べる．

改良 1 のプログラム (再帰型)

```
#include<stdio.h>
int fibo(int n)
{
if(n < 2) {return (n);}
else {return (fibo(n-1)+fibo(n-2));}
}
```

改良 2 のプログラム (再帰型)

```
#include<stdio.h>
int fibo(int n)
{
return n>2 ? fibo(n-1)+fibo(n-2):1;
}
```

改良 3 のプログラム (再帰型)

```
#include <stdio.h>
int fibo(n)
int n;
{ switch (n) {
case 0:     return 0;
case 1:
case 2:     return 1;
case 3:     return 2;
case 4:     return 3;
default:
return fibo(n - 1)+fibo(n - 2);
}}
```

改良 4 のプログラム (再帰型)

```
#include <stdio.h>
#define DATA_MAX 15
int data[DATA_MAX] = { 0, 1 };
int data_size = 2;
int fibo(n)
int n;
{ if (n >= data_size) {
return fibo(n-1)+fibo(n - 2);
} else {
return data[n];
}}
main()
{ int n, r;
long t1, t2;

printf("n = ? ");
scanf("%d", &n);
```

```
time(&t1);
/* make initial data */
for (data_size = 2; data_size
        < DATA_MAX; data_size++) {
data[data_size] = fibo(data_size);
}
r = fibo(n);
time(&t2);
printf("fibo(%d)=%d\n", n, r);
printf("time=%ld sec.\n", t2 - t1);
exit(0);}
```

一方，繰返し型のプログラムのうち，Table 2 で節約版と呼んだのは，作業領域:w を消去した以下のプログラムをさす [2].

節約版のプログラム (繰返し型)

```
function fibo(i:integer):integer;
 var a,b,w,n:integer;
begin
 a:=0; b:=1;n:=1;
 while n<> i do begin
 a:=a+b; (a に a+b の値を入れる)
 b:=a-b; (b にもとの a の値を入れ直す)
 n:=n+1
 end;
 fibo:=a
end;
```

なお，この方法を独力で考えついた学生は，全体の 20%弱であった.

3 組合せの数: ${}_nC_r$の計算

ここでは，再帰型の例題としてよく採用される階乗 ($n!$) の計算方法について考える [6] [9] [11]. 関係 $n! = n \cdot (n-1)!$, $0! = 1$ を PASCAL で記述すると次のように表せる.

```
function fac(n:integer):integer;
begin
     if n=0 then fac:=1
     else fac:=n*fac(n-1)
end;
```

しかし，このままのプログラムでは，計算機で 1 word を使って表現できる数の制限をすぐ越えてしまう. 例えば，C 言語で int 型で表現できる最大の数は，$2^{31}-1 = 2147483647$ であり，12! で数の限度をオーバーしてしまう. 同様に，組合せの数:${}_nC_r$を求める問題も事情は同じである. 階乗の計算を定義通り行うと，${}_{17}C_8$でやはり制限を越えてしまう. もちろん，${}_{17}C_8$=17!/(8!9!) の計算は分子·分母を 9!で割って計算するものとする (参考までに，分子$\approx 9 \times 10^9$, 分母$\approx 4 \times 10^4$).

$$ {}_nC_r = \frac{n(n-1)\cdots\{n-(r-1)\}}{r\cdot(r-1)\cdots 1} $$

しかし，この場合には上の式を注意深く観察し工夫すれば，もっと大きな数まで計算することができる. そこで学生に与えた課題は次の通りである.

> [**課題 2**]: 組合せの数:${}_nC_r$ が求められる (出来るだけ大きな数 n まで計算出来るように工夫した) プログラムを作成せよ. また，計算できる最大の n と${}_nC_r$の値を示せ. ただし，r=$\frac{n}{2}$または r=$\frac{n-1}{2}$とする.

この**課題 2** に対する解答結果を Table 3 に示す. 全体の 86%の学生が正解に達した. n=33 が整数 int 型の正解，n=55 が倍精度実数 double 型の正解，そして，それ以上大きな n まで求めた解答は，1 word ではなく長大桁を扱えるように配列を使って工夫した結果である.

Table 3. Student's results for calculation of Combination: ${}_nC_r$.

n=33	n=55	n≈200	$n \geq 10^4$	others
61%	9%	5%	11%	14%

例えば，${}_{100000}C_{50000}$, ${}_{200000}C_{100000}$を求めたときの計算時間は各々約19分40秒と1時間20分であった(前者ぐらいが他の学生に迷惑を及ぼさない学生実習の限界)．前者の桁数は30101桁になった．また，長大桁計算の概略は次の通りである．例えば，${}_{20}C_{15}$の場合を考える．

$$ {}_{20}C_{15} = \frac{20!}{5!\,15!} = \frac{20 \cdot 19 \cdot 18 \cdot 17 \cdot 16}{5 \cdot 4 \cdot 3 \cdot 2 \cdot 1} $$

ここで，分子と分母を各々α, βとおくと，

$$ \begin{aligned} \alpha &= (2^2 \times 5) \cdot 19 \cdot (2 \times 3^2) \cdot 17 \cdot 2^4 \\ &= 2^7 \times 3^2 \times 5^1 \times 17^1 \times 19^1 \\ \beta &= 5 \times 2^2 \times 3 \times 2 = 2^3 \times 3^1 \times 5^1 \\ \frac{1}{\beta} &= 2^{-3} \times 3^{-1} \times 5^{-1} \end{aligned} $$

したがって，

$$ {}_{20}C_{15} = 2^4 \times 3^1 \times 17^1 \times 19^1 $$

と素数の積に分解でき，${}_nC_r$(=分子/分母)の計算は，最終的に素数の指数の加算と減算に帰着できる．そして，最後に残った各素数を掛け合わせればよい．さらに，桁あふれを防止しすべての桁を表示するために，必要な桁数分の配列を用意し，最終的な結果を1つ1つの配列に収める工夫を施す必要がある．

4 積和計算とコンパイラ

ここでは，学生(情報工学科3年生)に次の課題を与えた．学生は，すでにFortranでは行方向，C言語では列方向にデータを参照すれば，メモリ上の記憶順序との関係から効率がよい，ことを習得している．

> **[課題 3]**: メモリアクセスと効率
> $S_1 = \sum_{i=0}^{n} A_i^2$, $S_2 = \sum_{i=n}^{0} A_i^2$, $S_3 = \sum_{j=0}^{m}\sum_{i=0}^{m} B_{ij}^2$, $S_4 = \sum_{i=0}^{m}\sum_{j=0}^{m} B_{ij}^2$ を求めるプログラムを作成し，CPU時間を比較せよ．ただし，n=10000, $m \doteq \sqrt{10000}$とする．また，配列A_i, B_{ij}には，sin関数の値を与える．

測定された各計算時間について，S_1を1としたときの比率をTable 4に示す．

Table 4. Ratios of CPU time for Product-Sum S_1, S_2, S_3 and S_4.

S_1	S_2	S_3	S_4
1.00	0.95	1.61	1.18

C言語では，列方向に計算するS_4の方が行方向のS_3に比べて計算が速いのは当然としても，S_1よりS_2の方が速いのが不思議である(これに実験後気付いた)．そこで，次の課題をさせる事態になった．

> **[課題 4]**:
> 課題3のプログラムを，オプション-Sを使ってコンパイルし，計測ループの部分に対してコンパイラーが出力したマシン命令を表5のようにまとめ，計算効率について考察せよ．

Table 5. Operation counts of Assembler coding for S_1 and S_2.

	命令数	メモリ	演算	その他
S_1	32	11	5	16
S_2	30	11	5	14

表中で，メモリとはload, store命令であり，演算とはadd, sub, mulなどの命令であ

り，その他には分岐命令やジャンプ命令などが含まれる．

図1に，2つの積和計算:S_1とS_2の計算フローの概略を示す．ただし，nは繰り返しの最大数である．図の中で，R*はレジスターを意味し，R0は0番目のレジスターを意味する．ただし，0番目のレジスターは常に値が0である．したがって，S_2の場合，終了回数をレジスターにセットする命令がloopの外に出せ，その分計算が速くなる．命令数だけで単純に比較すれば，S_1の場合が32個，S_2の場合が30になり，その割合は約7%になり，表4の差が生じたことになる．

Fig. 1. Schema of Assembler Coding for product-sum S_1 and S_2.

(a)$S_1 = \sum_{i=0}^{n} A_i^2$

	i=0 (初期値設定)
loop	R* = n
	i > R* ⟹ 終了
	i = i+1
	<積和の計算>
	loop の最初に戻る

(b)$S_2 = \sum_{i=n}^{0} A_i^2$

	i=n (初期値設定)
loop	i < R0(=0) ⟹ 終了
	i = i-1
	<積和の計算>
	loop の最初に戻る

減少した2つの命令とは，SPARCでは次のものである．

sethi %hi(value), %o1
or %o1, %lo(value), %o2

最初の行で，即値22bitsをレジスター%o1の上位に，Zero 10bitsをレジスター%o1の下位に入れる．そして，次の命令は，レジスター%o1の値と，符号拡張された13bits即値でORを取り，結果をレジスター%o2に入れることを意味する．

5 おわりに

再帰型のプログラミングに関する実践報告をもとに，H.P.C.の立場からいくつかの話題を提供した．

謝辞 日頃，HPCに関して討論いただく弘中助教授に感謝の意を表したい．また，本発表に対して，協力してくれた情報工学科2年生と3年生全員に感謝したい．

参考文献

[1] 小川貴英, LISPプログラミング入門, 岩波書店, 1989, p.80.

[2] 奥村晴彦, アルゴリズム事典, 技術評論社, 1994, pp.59–60.

[3] 駒木悠二, 有澤誠編, ナノピコ教室, 共立出版, 1990, pp.146–150.

[4] 寒川光, RISC超高速化プログラミング技法, 共立出版, 1995.

[5] 島内剛一他編, アルゴリズム辞典, 共立出版, 1994, p.658.

[6] 波平博人, アルゴリズム, CQ出版社, 1995, pp.151–154.

[7] 萩原兼一, 基礎 PASCAL, 岩波書店, 1993, pp.127–130.

[8] Makino, J., "Lagged-Fibonacci random number generators on parallel computers", Parallel Computing, **20**(1994), pp.1357–1367.

[9] 森口繁一, 小林光夫, 武市正人, Pascalプログラミング対話, 共立出版, 1993, pp.173–175.

[10] Petersen, W.P., Lagged Fibonacci Series Random Number Generators for the NEC SX-3, Int. J. of High Speed Computing, **6**(1994), pp.387–398.

[11] 渡辺 宏, LISPプログラミング, 共立出版, 1988, pp.49–53.