「精度保証数値計算」 レポート課題

学籍番号:600P064-9　氏名:吉田 直史　協力:岡部 貴弘
理工学研究科　情報科学専攻

演習 5-2

Scilab の ATLAS (or 最適化 BLAS) による高速化を行え.

解答

1. ATLAS とは何か

ATLAS(Automatically Tuned Linear Algebra Software) は BLAS と LAPACK の一部をそれらを走らせるマシンに最適化されたものとして生成するフリーのソフトウェアで, そのソースファイルは http://www.cs.utk.edu/ rwhaley/ATLAS の /OS/index.html からダウンロードできる.

2. ATLAS による Octave の高速化

ATLAS による Scilab の高速化を行う前に, まずは, すでに実現されている ATLAS による Octave の高速化を行ってみることで, どのように ATLAS を Octave に組み込んでいるのか, また, ATLAS を組み込むことで実際にどの程度 Octave が高速化されるのかを見てみることにする.

2.1 ATLAS のインストール

ダウンロードした ATLAS のソースファイル (atlas3.1.6D.tgz) を展開し, 以下のようにコンパイルする.

```
tar zxf atlas3.1.6D.tgz
```

```
cd ~/ATLAS
gmake
gmake install arch=Linux_PIII
```

以上により, ~/ATLAS/lib/Linux_PIII/ 内に 5 つのライブラリ libatlas.a, libcblas.a, libf77blas.a, liblapack.a, libtstatlas.a が作成される. ここでコンパイルに時間がかかるのは, 現在使用しているマシンに対して最適化された BLAS や LAPACK をこの段階で作成しているためであると考えられる.

ここで, Octave のコンパイルを行う.

```
tar zxf octave-2.0.16.tar.gz
cd ~/octave-2,0,16
sh ./configure
gmake FFLAGS=-02 CXXFLAGS=-02 CFLAGS=-02
```

次に, 以下のようにして ATLAS を Octave に組み込む.

```
mkdir tmp
cd tmp
ar x ~/ATLAS/lib/Linux_PIII/libatlas.a
ar x ~/ATLAS/lib/Linux_PIII/libcblas.a
ar x ~/ATLAS/lib/Linux_PIII/libf77blas.a
ar x ~/ATLAS/lib/Linux_PIII/liblapack.a
ar x ~/ATLAS/lib/Linux_PIII/libtstatlas.a
ar r ~/octave-2.0.16/libcruft/libcruft.a *.o ((注) a と * の間は半角1文字分の空白を空ける)
cd ..
rm -r tmp
```

ここでは, 先の ATLAS のコンパイルによって作成された 5 つのライブラリファイル *.a を, ar x コマンドによって一度それぞれのライブラリを構成している各オブジェクトファイル *.o に分解し, ar r コマンドによって, それらのオブジェクトファイルの中で ~/octave-2.0.16/libcruft/ の中のライブラリファイル libcruft.a に含まれていないものを libcruft.a に付け加えるという作業をしている.

最後に, Octave を次のように再構築する.

```
cd ~/octave-2.0.16/libcruft/lapack/
g77 -02 -c ilaenv.f
cd ../../
gmake
gmake install
```

ここでは, まず ~/octave-2.0.16/libcruft/lapack/ の中の ilaenv.f を一度コンパイルしてから, 必要なライブラリを作りなおし Octave をインストールしなおしている.

以上で, Octave に ATLAS が組み込まれた. 実行環境は以下の通りである.

実行環境

CPU : Mobile Intel Celeron 500MHz

OS : Vine Linux 2.0

2.2 実行速度の比較

以下のようにして, 行列の乗算, LU 分解, QR 分解, 連立一次方程式, 固有値を計算するときの実行時間を求めた.

1. 行列の乗算

```
octave:> A=rand(1000,1000);
octave:> B=rand(1000,1000);
octave:> tic;A*B;toc
```

2.LU 分解

```
octave:> A=rand(1000,1000);
octave:> tic;[L U P]=lu(A);toc
```

3.QR 分解

```
octave:> A=rand(1000,1000);
octave:> tic;[Q R P]=qr(A);toc
```

4. 連立一次方程式

```
octave:> A=rand(1000,1000);
octave:> b=rand(1000,1);
octave:> tic;x=A\b;toc
```

5. 固有値

```
octave:> A=rand(1000,1000);
octave:> tic;eig(A);toc
```

ATLAS を組み込む前後で Octave の実行速度を比較すると以下の表のようになる. 実行速度は, 5 回実行を行いその平均をとった.

	ATLAS 組込み前	ATLAS 組込み後
1. 行列の乗算	56.839 (sec)	5.5826 (sec)
2.LU 分解	22.635 (sec)	5.7345 (sec)
3.QR 分解	71.297 (sec)	31.657 (sec)
4. 連立一次方程式	18.642 (sec)	15.648 (sec)
5. 固有値	297.87 (sec)	269.07 (sec)

以上の結果から, ATLAS を組み込んだ Octave は組み込む前に比べていずれの計算も高速化されていることがわかった. とくに, 行列の乗算では 10 倍, LU 分解では 4 倍もの高速化が実現できた.

3. ATLAS による Scilab の高速化

ここまでに行った ATLAS による Octave の高速化を参考にして, ATLAS による Scilab の高速化を試みた.

3.1 新しい LAPACK の導入

Scilab に入っている LAPACK は, 最小限のファイルしか入っていないためか, ファイルサイズが小さい. そこで, 完全な LAPACK を使用するために, `http://phase.etl.go.jp/mirrors/netlib/lapack/index.html` から lapack.tgz をダウンロードする.

まず, lapack.tgz を展開すると `~/LAPACK/` というディレクトリができる. ここで, LAPACK を Linux でインストールするために, `~/LAPACK/INSTALL/` の中の make.inc.LINUX というファイルを make.inc として `~/LAPACK/` 中の make.inc と置き換える必要がある. make.inc を Linux 用に置き換えたら, make all を行う. すると, `lapack_LINUX.a` という LAPACK のライブラリができるので, このファイルを lapack2.a という名前に変えて `~/scilab-2.5/libs/` に置く.

ここで, ATLAS で最適化した LAPACK のライブラリで, `~/ATLAS/lib/Linux_PIII/` の中にある liblapack.a を

```
mkdir tmp
cd tmp
ar x ~/ATLAS/lib/Linux_PIII/liblapack.a
ar r ~/scilab-2.5/libs/lapack2.a *.o
cd ..
rm -r tmp
```

として, lapack2.a に加える. これで, 最適化された完全な LAPACK のライブラリ lapack2.a

ができた. また,ATLASで最適化した残りのライブラリ libatlas.a, libcblas.a, libf77blas.a, libtstatlas.a も ~/scilab/libs に置く.

ここで, Makefile.OBJ.in の内容を書き換える. 最適化する前の LAPACK および BLAS のライブラリを記述している部分 $(SCIDIR)/libs/lapack.a と $(SCIDIR)/libs/blas.a を消して, 代わりに, 以下のように ATLAS で最適化したライブラリを付け加える.

```
LIBRSCI = $(SCIDIR)/libs/system.a $(SCIDIR)/libs/interf.a \
        $(SCIDIR)/libs/system2.a $(SCIDIR)/libs/optim.a \
        $(SCIDIR)/libs/integ.a $(SCIDIR)/libs/control.a \
        $(SCIDIR)/libs/scicos.a $(SCIDIR)/libs/signal.a \
        $(SCIDIR)/libs/poly.a $(SCIDIR)/libs/calelm.a \
        $(SCIDIR)/libs/lapack2.a $(SCIDIR)/libs/graphics.a \
        $(SCIDIR)/libs/sparse.a $(SCIDIR)/libs/metanet.a \
        $(SCIDIR)/libs/sun.a $(SCIDIR)/libs/gd.a  \
        $(SCIDIR)/libs/intersci.a  $(SCIDIR)/libs/@GUILIB@.a \
        $(SCIDIR)/libs/graphics.a $(SCIDIR)/libs/menusX.a \
        $(SCIDIR)/libs/libcomm.a $(SCIDIR)/libs/comm.a \
        $(SCIDIR)/libs/sound.a $(SCIDIR)/libs/dcd.a $(SCIDIR)/libs/rand.a \
        $(SCIDIR)/libs/libf77blas.a $(SCIDIR)/libs/libcblas.a \
        $(SCIDIR)/libs/libatlas.a $(SCIDIR)/libs/libtstatlas.a \
        $(SCIDIR)/libs/fraclab.a $(SCIDIR)/libs/int.a \
@PVMSCILIB@ @XDRLIBNAME@ @TKSCILIB@
```

のように書き換え, ./configure および make all を行って, Scilab を再構築する.

以上で, 最適化された LAPACK および BLAS が組み込まれた Scilab を再構築することができた. しかし, この Scilab を使って行列の積や, LU 分解, QR 分解などを行ってみたが, いすれも高速化されていなかった. そこで, LU 分解に的をしぼって, なぜ高速化ができないのかを調べてみた.

3.2 LU 分解の高速化

LAPACK で LU 分解について記述しているのは

```
~/scilab-2.5/routines/lapack/dgetrf.f
```

というファイルて定義されている dgetrf という関数ある. しかし, 実際に Scilab で LU 分解を行っているのは

```
~/scilab-2.5/routines/interf/matlu.f
```

に組み込まれている

```
~/scilab-2.5/routines/control/dgefa.f
```

というファイルで定義されている dgefa という関数で, この関数は linpack を用いているだけで, LAPACK の LU 分解の関数 dgetrf を使っていない. LAPACK がいくら最適化されたものであっても, その最適化された関数を使っていないのであれば, LU 分解が高速化されることはない.

そこで, linpack を用いている関数 dgefa から LAPACK 中の dgetrf を呼び出す方法を考えた. dgefa.f ははじめ以下のように記述されていて, LAPACK を用いていない.

```
c--------------- dgefa.f(original) ---------------

      subroutine dgefa(a,lda,n,ipvt,info)
      integer lda,n,ipvt(*),info
      double precision a(lda,*)

      double precision t
      integer idamax,j,k,kp1,l,nm1

      info = 0
      nm1 = n - 1
      if (nm1 .lt. 1) go to 70
      do 60 k = 1, nm1
         kp1 = k + 1

         l = idamax(n-k+1,a(k,k),1) + k - 1
         ipvt(k) = l

         if (a(l,k) .eq. 0.0d+0) go to 40

            if (l .eq. k) go to 10
               t = a(l,k)
               a(l,k) = a(k,k)
               a(k,k) = t
   10       continue

            t = -1.0d+0/a(k,k)
            call dscal(n-k,t,a(k+1,k),1)
```

```
            do 30 j = kp1, n
               t = a(l,j)
               if (l .eq. k) go to 20
                  a(l,j) = a(k,j)
                  a(k,j) = t
   20          continue
               call daxpy(n-k,t,a(k+1,k),1,a(k+1,j),1)
   30       continue
         go to 50
   40    continue
            info = k
   50    continue
   60 continue
   70 continue
      ipvt(n) = n
      if (a(n,n) .eq. 0.0d+0) info = n
      return
      end

c--------------------------------------------------
```

そこで, この関数 dgefa の LU 分解に関する記述を消して, LAPACK の関数 dgetrf を呼び出すように書き換えた. ここで, 2 つの関数の引数の並びや内容が少し違うので, そのことに注意しなければならない. 関数 dgetrf は m × n 行列を扱えるが, 関数 dgefa は n × n の正則行列しか扱えないので, dgetrf(m,n,···) を dgetrf(n,n,···) と記述する. また, 書き換えた後, dgefa.f を `g77 -c dgefa.f` でコンパイルして dgefa.o を作り直す必要がある.

```
c--------------- dgefa.f(type2) ---------------

      subroutine dgefa(a,lda,n,ipvt,info)
      integer lda,n,ipvt(*),info
      double precision a(lda,*)
      call dgetrf(n,n,a,lda,ipvt,info)
      return
      end

c--------------------------------------------------
```

これで, 再度 ./configure および make all し Scilab を再構築すると, LU 分解の計算速度に関しては高速化が実現できた. しかし, 関数 dgefa と dgetrf の返す値が違うため, LU 分解の計算結果の l の値が正しく出ない. そこで, l の値を正しく求めるように, dgefa の記述をさらに以下のように書き加えた.

```
c--------------- dgefa.f(type3) ---------------

      subroutine dgefa(a,lda,n,ipvt,info)
      integer lda,n,ipvt(*),info
      double precision a(lda,*)

      integer kb, k, i

      call dgetrf(n,n,a,lda,ipvt,info)

      do 12 kb = 1, n
c     do 12 kb = n, 1, -1
         k = n+1-kb
         if(k.ne.n) then
            call dscal(n-k,-1.0d+0,a(k+1,k),1)
         endif
         i = ipvt(k)
         if (i.ne.k .and. k.ne.0) then
            call dswap(k-1,a(i,1),lda,a(k,1),lda)
         endif
 12   continue

      if (info.lt.0) then
         info = -info
      endif

      return
      end

c-------------------------------------------------
```

これで, ./configure および make all し Scilab を再構築すると, 高速化された, 正しい LU 分解が実現できた. 以上のようにすれば, 最適化された LAPACK の関数を用いた高速

化された LU 分解が可能になる.

3.3 実行結果

(1) 5×5 行列を与え,それを LU 分解した結果を, dgefa(original), dgefa(type2), dgefa(type3) の 3 つをそれぞれ使用した場合で比較する.

```
----- original -----------------------------------
-->a=[1 2 3 4 5;2 3 4 5 6;3 4 5 6 7;4 5 6 7 8;5 6 7 8 9]
 a  =

!   1.    2.    3.    4.    5. !
!   2.    3.    4.    5.    6. !
!   3.    4.    5.    6.    7. !
!   4.    5.    6.    7.    8. !
!   5.    6.    7.    8.    9. !

-->a=[1 2 3 4 5;2 3 4 5 6;3 4 5 6 7;4 5 6 7 8;5 6 7 8 9];

-->[l u p]=lu(a)
 p  =

!   0.    0.    0.    0.    1. !
!   1.    0.    0.    0.    0. !
!   0.    0.    1.    0.    0. !
!   0.    1.    0.    0.    0. !
!   0.    0.    0.    1.    0. !
 u  =

!   5.    6.     7.     8.     9.  !
!   0.    0.8    1.6    2.4    3.2 !
!   0.    0.     0.     0.     0.  !
!   0.    0.     0.     0.     0.  !
!   0.    0.     0.     0.     0.  !
 l  =

!   1.     0.      0.           0.    0. !
!   0.2    1.      0.           0.    0. !
```

```
!   0.6    0.5     1.           0.    0. !
!   0.4    0.75    0.3695998    1.    0. !
!   0.8    0.25    0.5          0.    1. !

-->p*a-l*u
 ans  =

!   0.            0.    0.    0.    0. !
!   0.            0.    0.    0.    0. !
! - 4.441E-16     0.    0.    0.    0. !
!   0.            0.    0.    0.    0. !
!   0.            0.    0.    0.    0. !
-------------------------------------------------

----- type2 ---------------------------------------
-->a=[1 2 3 4 5;2 3 4 5 6;3 4 5 6 7;4 5 6 7 8;5 6 7 8 9]
 a  =

!   1.    2.    3.    4.    5. !
!   2.    3.    4.    5.    6. !
!   3.    4.    5.    6.    7. !
!   4.    5.    6.    7.    8. !
!   5.    6.    7.    8.    9. !

-->[l u p]=lu(a)
 p  =

!   0.    0.    0.    0.    1. !
!   1.    0.    0.    0.    0. !
!   0.    0.    1.    0.    0. !
!   0.    1.    0.    0.    0. !
!   0.    0.    0.    1.    0. !
 u  =

!   5.    6.     7.     8.     9.  !
!   0.    0.8    1.6    2.4    3.2 !
!   0.    0.     0.     0.     0.  !
```

```
!   0.    0.     0.     0.     0.  !
!   0.    0.     0.     0.     0.  !
 l  =

!   1.     0.      0.           0.    0. !
! - 0.8    1.      0.           0.    0. !
! - 0.6  - 0.5     1.           0.    0. !
! - 0.2  - 0.25  - 0.5          1.    0. !
! - 0.4  - 0.75  - 0.5869865    0.    1. !

-->p*a-l*u
 ans  =

!   0.    0.     0.     0.     0.  !
!   5.    6.     7.     8.     9.  !
!   6.    8.     10.    12.    14. !
!   3.    4.4    5.8    7.2    8.6 !
!   6.    8.     10.    12.    14. !
-------------------------------------------------

----- type3 ---------------------------------------
-->a=[1 2 3 4 5;2 3 4 5 6;3 4 5 6 7;4 5 6 7 8;5 6 7 8 9]
 a  =

!   1.    2.    3.    4.    5. !
!   2.    3.    4.    5.    6. !
!   3.    4.    5.    6.    7. !
!   4.    5.    6.    7.    8. !
!   5.    6.    7.    8.    9. !

-->[l u p]=lu(a)
 p  =

!   0.    0.    0.    0.    1. !
!   1.    0.    0.    0.    0. !
!   0.    0.    1.    0.    0. !
!   0.    1.    0.    0.    0. !
```

```
!   0.    0.    0.    1.    0. !

 u  =

!   5.    6.     7.     8.     9.  !
!   0.    0.8    1.6    2.4    3.2 !
!   0.    0.     0.     0.     0.  !
!   0.    0.     0.     0.     0.  !
!   0.    0.     0.     0.     0.  !
 l  =

!   1.     0.      0.           0.    0. !
!   0.2    1.      0.           0.    0. !
!   0.6    0.5     1.           0.    0. !
!   0.4    0.75    0.5869865    1.    0. !
!   0.8    0.25    0.5          0.    1. !

-->p*a-l*u
 ans  =

!   0.           0.    0.    0.    0. !
!   0.           0.    0.    0.    0. !
! - 4.441E-16    0.    0.    0.    0. !
!   0.           0.    0.    0.    0. !
!   0.           0.    0.    0.    0. !
-------------------------------------------------
```

この結果から, p , u の値は 3 つの場合とも等しいが, l の値は type2 を用いたときだけ他と違って負の値が出ている. また, type2 では, `p*a-l*u` の値が 0 にならないことから type2 は LU 分解が正しくできていない. 一方, `p*a-l*u` の値が限りなく 0 に近いことから original と type3 では LU 分解が正しくできていることがわかる.

(2) 1000 × 1000 行列を与え,それを LU 分解したときにかかる計算時間を, dgefa(original), dgefa(type2), dgefa(type3) の 3 つをそれぞれ使用した場合で比較する.

```
-->stacksize(10000000);

-->a=rand(1000,1000);
```

```
-->timer();[l u p]=lu(a);timer()
```

dgefa.f	original	type2	type3
time(sec)	17.98	4.74	4.76

この結果から, LU 分解の計算速度は original と比較して, type2, type3 では 4 倍近くも高速化されていることがわかる.

(3) 連立一次方程式を解くときにかかる計算時間を, dgefa(original), dgefa(type2), dgefa(type3) の 3 つをそれぞれ使用した場合で比較する.

```
-->stacksize(10000000);

-->A=rand(1000,1000);

-->b=rand(1000,1);

-->timer();A\b;timer()
```

dgefa.f	original	type2	type3
time(sec)	15.56	2.42	2.54

この結果から, 連立一次方程式は Octave の高速化のときよりもはるかに高速化されていることがわかる, これは Scilab の連立一次方程式では, LU 分解の関数を使用しているためであると考えられる.

実行環境

CPU : Mobile Intel Celeron 500MHz
OS : Vine Linux 2.0

3.4 結論

以上の結果から, LU 分解と連立一次方程式に関してのみであるが, Scilab を高速化することができた. 同様に, 高速化できていない命令では, 最適化された LAPACK や BLAS の

関数を使っていない可能性が高い. よって, 高速化できていない命令を高速化するためには, 最適化された LAPACK や BLAS の関数を使うようにソースコードを書き直す必要があると考えられる.

参考文献

[1] 大石進一著: “Linux 数値計算ツール”, コロナ社 (2000).