原子核構造の理論的計算におけるベクトル化の試み

大阪大学理学部　田　村　圭　介

1. はじめに

近年、我々を取り巻く計算機の環境は目まぐるしく変化している。特に、パーソナル・コンピュータの攻勢は目を見張るものがあり、数年前までは、夢であったような事が、次々に机の上に乗るようなシステムの中で現実のものになっている。我々の研究室では、プログラムを作るときは、まずパソコンの前に座り、スイッチを入れて…… というのが、よく見かけられるようになってきている。更に、サブルーチンごとのチェックもそのままoff-line で済ませ、ほぼ完成した形で大型計算機に移すことが出来るようになってきている。最近では、本当にパソコンなのか？と思えるほどの記憶容量と処理速度を持ったものも発表されている。この様なシステムが安価で個人の手にはいるようになると、「ぜひ、大型計算機でなければならない」ということも少なくなっていくように思える。しかし、高速処理、大記憶容量を必要とする計算は、大型機にたよるしかなく、「より速い、より大きい」計算機が要求されることになる。私の専攻する原子核理論物理では、議論が精密化されるに従って、長大なＣＰＵ時間を必要とする数値計算が要求される。こうなってくると計算にかかる金額が非常に重要になってきて、本質的でない問題で悩まなければならないことになってしまう。（昨年秋のＳＸ－１無料期間中のＪＯＢのこみ方から想像して、大勢の方々が同様の悩みを抱えておられるのでしょう。） これまで、Ｓ1000 上で開発されてきたＬＩＢＲＡＲＹが、ＳＸ－１によってどの程度高速化され、我々の悩みを解決してくれるのか？ということを調べるために行った試みを報告する。

2. ベクトル化の試み

原子核構造の理論的研究においては、回転群についての代数的計算、ベッセル関数、ルジャンドル関数等の多くの特殊関数を含む積分を実行しなければならない。今回は、これらのうち、回転群代数における６－Ｊ係数、９－Ｊ係数、及び球ベッセル関数について報告を行う。

６－Ｊ係数は、以下の式で定義される[1)]。

$$\begin{Bmatrix} j_1 & j_2 & j_3 \\ L_1 & L_2 & L_3 \end{Bmatrix} = \triangle(j_1 j_2 j_3)\,\triangle(j_1 L_2 L_3)\,\triangle(L_1 j_2 L_3)\,\triangle(L_1 L_2 j_3)$$
$$\times \sum_K (-)^K (K+1)!\,[(K-j_1-j_2-j_3)!\,(K-j_1-L_2-L_3)!\,(K-L_1-j_2-L_3)!$$
$$\times (K-L_1-L_2-j_3)!\,(j_1+j_2+L_1+L_2-K)!\,(j_2+j_3+L_2+L_3-K)!\,(j_1+j_3+L_3+L_1-K)!\,]^{-1}$$

$$\triangle(abc) = [\,(a+b-c)!\,(a-b+c)!\,(-a+b+c)!/(a+b+c+1)!\,]^{1/2}$$

これに対するコードを（Ｐ１）に与える。△（ａｂｃ）、binomial、及び$(-)^K$を配列に貯めることによって高速化を計った。（始めにSUBROUTINE　INITをCALLする。）kについてのloopはベクトル化されている。

表 1

引数	0	2	4	6	8	10
CP　（μsec）	9.5	10.7	11.8	12.9	14.1	15.2
AP　（μsec）	5.7	5.9	5.8	6.0	5.7	5.7
N. V.（μsec）	4.3	5.0	5.7	6.4	7.1	7.8
CP/AP	1.7	1.8	2.0	2.2	2.5	2.7

$j_1 = j_2 = \cdots = L_3 = 0 \sim 10$に対して計算に要した時間をCP, APについて比較してみると（表１）の様になる。ベクトル化の効果（AP）は明白で、引数の値が増加しても計算時間は変化しない。ただし、引数が０～４の場合は、loopの繰り返し回数が５回以下となり、NOVECTOR指示をした方が高速となる。（N.V.）

９－Ｊ係数は、６－Ｊ係数によって以下のように定義される[1)]。

$$\begin{Bmatrix} a & b & c \\ c & d & e' \\ f & f' & g \end{Bmatrix} = (-)^{2(a+e'+f')} \sum_{\lambda} (2\lambda+1) \begin{Bmatrix} b & e & a \\ c & f & \lambda \end{Bmatrix} \begin{Bmatrix} e & g & e' \\ d & c & \lambda \end{Bmatrix} \begin{Bmatrix} f & g & f' \\ d & b & \lambda \end{Bmatrix}$$

表 2

引数	2	4	6	8	10
CP　（msec）	0.17	0.14	0.21	0.29	0.38
AP　（msec）	0.07	0.10	0.14	0.18	0.22
CP/AP	2.4	1.4	1.5	1.6	1.7

プログラムには、６－Ｊ係数に相当するloopとその外にλについてのloopがある。（Ｐ２）

２重loop構造になっているため、内側のloopのみしかベクトル化の対象にならない。６－Ｊ係数ほど顕著ではないが、引数が大きくなると、ベクトル化による効果が現れる。（表２）

球ベッセル関数$j_\ell(x)$は、十分に大きなℓに対して、適当に小さな数値を与え、漸化式、及びsumrule

$$j_\ell(x) = \frac{2\ell-1}{x} j_{\ell-1}(x) - j_{\ell-2}(x)$$

$$\sum_{\ell=0}^{\infty} (2\ell+1)\, j_\ell(x)^2 = 1$$

表 3

L＼X		2	4	6	8	10
	AP (μsec)	31.4	33.9	35.2	35.8	36.5
0	CP (μsec)	41.4	58.9	55.5	58.1	62.4
	CP/AP	1.3	1.7	1.6	1.6	1.7
	AP	32.6	33.6	35.2	35.8	36.8
2	CP	44.5	49.5	54.8	58.4	62.4
	CP/AP	1.4	1.5	1.6	1.6	1.7
	AP	32.7	34.0	35.2	35.5	36.8
4	CP	46.9	49.9	55.1	58.2	63.4
	CP/AP	1.4	1.5	1.6	1.6	1.7

によって、要求された次数の関数値を得るというアルゴリズムを採用している。（Ｐ３）３項間の漸化式を計算する部分がベクトル化の対象外となる。そのため、作業用の配列を作り、可能な限りベクトル化されるように loop を分解した。（表３）で見られるように、ｘが大きくなるに従って、CP と AP での違いが現れる。このようなアルゴリズムによると、必要であればあるＸについての $\ell = 0$ から十分に大きな ℓ について、球ベッセル関数の値を一度に得ることが可能になる。

3. おわりに

我々の研究分野で必要となる数値計算では、ベクトルプロセッサーによって、飛躍的に高速化されるものは多くない。今回対象にした function は、我々の研究分野では多用されるものであり、数値計算の多くの場合律速段階となるものである。結果的には飛躍的な処理速度の向上はみられなかったが、今回行ったように、個々の loop についてこまめに検討し、僅かづつでも各部分を高速化していくことが、全体の処理速度の向上につながっていくと思われる。

1) EDOMONS, A. R. ANGULAR MOMENTUM IN QUANTUM MECHANICS, PRINCETON UNIVERSITY PRESS

(P1)

```
c--------+---------+---------+---------+---------+---------+---------+
      double precision function f6j(j1,j2,j3,L1,L2,L3)
c--------+---------+---------+---------+---------+---------+---------+
      implicit double precision (a-h,o-z)
      parameter ( is=50 )
      common / fact / b(0:is,0:is),s(0:is),d(0:is,0:is,0:is)
c
      f6j = 0
c
      nmax = min0(j2+j3+L2+L3,j3+j1+L3+L1,j1+j2+L1+L2)/2
      if ( nmax > is ) return
c
      m0 = ( j1+j2+j3 )/2
      n1 = ( j1+L2+L3 )/2
      n2 = ( L1+j2+L3 )/2
      n3 = ( L1+L2+j3 )/2
      nmin = max0( m0,n1,n2,n3 )
      if ( nmin > nmax ) return
      t  = d(n1,j1,L2)*d(n2,L1,j2)*d(n3,L1,L2)
      tt = d(m0,j1,j2)
c
      if ( t.eq.0 .or. tt.eq.0 ) return
      t = sqrt(t/tt)
c
      m1  = m0-j1
      m2  = m0-j2
      m3  = m0-j3
      m0  = m0+1
      sum = 0
      do 100 k = nmin , nmax
 100  sum = sum + s(k)*b(m1,k-n1)*b(m2,k-n2)*b(m3,k-n3)*b(k+1,m0)
      f6j = t*sum
c
      end
```

(P2)

```
c--------+---------+---------+---------+---------+---------+---------+
      double precision function f9j( ia, ib , ie ,
     &                               ic, id , iep,
     &                               if, ifp, ig )
c--------+---------+---------+---------+---------+---------+---------+
      implicit double precision (a-h,o-z)
      parameter( is = 50 )
      common / fact / b(0:is,0:is),s(0:is),d(0:is,0:is,0:is)
      dimension Lzmax(5),Lzmin(5),mm(5),m(5),n(5),L(5)
c
      f9j = 0
c
      iacf  = (ia+ic+if )/2
      icdep = (ic+id+iep)/2
      ibdfp = (ib+id+ifp)/2
```

```
      iabe  = (ia+ib+ie )/2
      ieepg = (ie+ig+iep)/2
      iffpg = (if+ig+ifp)/2
c
      nbea  = iabe  - ia
      naeb  = iabe  - ib
      nabe  = iabe  - ie
      nepge = ieepg - ie
      neepg = ieepg - ig
      negep = ieepg - iep
      nfpgf = iffpg - if
      nffpg = iffpg - ig
      nfgfp = iffpg - ifp
c
      ibf   = ib + if
      ice   = ic + ie
      idg   = id + ig
c
      LLmax = min0( ib+if      , ic+ie      , id+ig       )
      LLmin = max0( iabs(ib-if), iabs(ic-ie), iabs(id-ig) )
      if (LLmin.gt.LLmax) return
c
      m (1) = min0( ie+if, ib+ic ) + ia
      m (2) = min0( ic+ig, id+ie ) + iep
      m (3) = min0( ib+ig, id+if ) + ifp
      mm(1) = ibf + ice
      mm(2) = ice + idg
      mm(3) = ibf + idg
      n (1) = max0( ibf  , ice   )
      n (2) = max0( ice  , idg   )
      n (3) = max0( ibf  , idg   )
      L (1) = max0( iabe , iacf  )
      L (2) = max0( icdep, ieepg )
      L (3) = max0( ibdfp, iffpg )
c
      t  = d(iabe,ia,ib)*d(ieepg,ie,ig)*d(iffpg,if,ig)
      tt = d(iacf,ia,ic)*d(icdep,ic,id)*d(ibdfp,ib,id)
      if ( t.eq.0.or.tt.eq.0 ) return
c
      t = dsqrt(tt/t)
      if (mod(ia+iep+ifp,2).eq.1) t=-t
c
      Lbfx = (ibf+LLmin)/2
      Lcex = (ice+LLmin)/2
      Ldgx = (idg+LLmin)/2
      sum = 0
      do 10 Lx = LLmin , LLmax , 2
        tt = dble(Lx+1)   *d(Lbfx,ib,if)
     &       *d(Lcex,ic,ie)*d(Ldgx,id,ig)
        do 20 k = 1 , 3
          Lzmax(k) = min0(  m(k)+Lx   , mm(K) )/2
          Lzmin(k) = max0( (n(k)+Lx)/2, L(k)  )
  20    continue
```

```
c
        sum1 = 0
        do 30 Lz1 = Lzmin(1) , Lzmax(1)
          sum1 = sum1 + s(Lz1)*b(naeb,Lz1-Lbfx)*b(nabe ,Lz1-Lcex)
     &                      *b(nbea,Lz1-iacf)*b(Lz1+1,iabe+1  )
  30    continue
c
        sum2 = 0
        do 40 Lz2 = Lzmin(2) , Lzmax(2)
          sum2 = sum2 + s(Lz2)*b(nepge,Lz2-Lcex )*b(neepg,Lz2-Ldgx)
     &                      *b(negep,Lz2-icdep)*b(Lz2+1,ieepg+1 )
  40    continue
c
        sum3 = 0
        do 50 Lz3 = Lzmin(3) , Lzmax(3)
          sum3 = sum3 + s(Lz3)*b(nfpgf,Lz3-Lbfx )*b(nffpg,Lz3-Ldgx)
     &                      *b(nfgfp,Lz3-ibdfp)*b(Lz3+1,iffpg+1 )
  50    continue
c
        sum = sum + tt*sum1*sum2*sum3
c
        Lbfx = Lbfx + 1
        Lcex = Lcex + 1
        Ldgx = Ldgx + 1
c
  10    continue
c
      f9j = sum * t
c
      end
c-------+---------+---------+---------+---------+---------+---------+
      subroutine init
c-------+---------+---------+---------+---------+---------+---------+
      implicit double precision (a-h,o-z)
      parameter( is = 50 )
      common / fact / b(0:is,0:is),s(0:is),d(0:is,0:is,0:is)
c
      ss = 1
      do 10 k = 0 , is
        s(k) = ss
        ss   = -ss
        do 10 j = 0 , is
          b(k,j) = 0
          do 10 L = 0 , is
            d6(k,j,L) = 0
            d9(k,j,L) = 0
  10  continue
c
      do 20 i = 0 , is
        b(i,0) = 1
        b(i,i) = 1
  20  continue
```

```
c
      do 30 i = 2 , is
        ks = 1
        do 30 j= 1 , i/2
          ks = ks*(i-j+1)
          ks = ks/j
          b(i,j  ) = ks
          b(i,i-j) = ks
  30  continue
c
      do 40 ia = 0 , is
      do 40 ib = 0 , is
        do 50 ic = iabs(ia-ib) , ia+ib , 2
          iabc = (ia+ib+ic)/2
          if ( iabc.gt.is ) go to 50
            d(iabc,ia,ib) = 1.d0/( dble(iabc+1)
     &                           *b(iabc,ia)*b(ia,iabc-ic) )
  50    continue
  40  continue
c
      end
```

(P3)

```
c--------+---------+---------+---------+---------+---------+---------+
      double precision function sbesl(L,x)
c--------+---------+---------+---------+---------+---------+---------+
      implicit double precision ( a-h,o-z )
      dimension s(-1:50),fx(50),ix(50)
c
      max     = 3.d0+3.d0*x**8.4d-2+9.d0*x**0.34d0+dmax1(dble(L),x)
      s(0)    = 1.d-50
      s(-1)   = 0
      sum     = s(0)*s(0)*dble(max+max+1)
      y       = 1.d0/x
      do 10 i = 1 , max
        n     = max - i
        n     = n+n+1
        fx(i) = (n+2)*y
  10    ix(i) = n
      do 20 i = 1 , max
  20    s(i)  = s(i-1)*fx(i) - s(i-2)
      do 30 i = 1 , max
  30    sum   = sum + s(i)*s(i)*ix(i)
      sbesl   = s(max-L)/sqrt(sum)
      end
```