$\mathcal{H}\Phi$におけるいくつかの工夫

三澤　貴宏

東京大学物性研究所計算物質科学研究センター

I.　はじめに

厳密対角化、とくに Lanczos 法を用いた大規模計算を行なう上での計算上の幾つかの工夫を解説する。行列要素が全て確保できない位巨大になってくると、行列要素を逐次生成しながら行列-ベクトル積の計算を行なうことになるが、その計算量は莫大になるので、計算コストを削減するために様々な工夫を行なうことになる。TITPACK ver.2[1] およびそのマニュアルはその工夫をどう行なうかを明確に解説してあり、その後の対角化のコードを作成する人にとって大きな参考となってきた。この解説では、TITPACK で用いている工夫に付け加えて、$\mathcal{H}\Phi$ で用いている工夫について解説する。

II.　記法などについて

この説明では、spin 1/2 の系を例にとって説明を行なう。S^z 成分を対角にする表示をとることで、Hilbert 空間の全ての要素を実空間配置にとることができる。例えば、$|\uparrow,\uparrow,\uparrow,\downarrow\rangle$ などは$\uparrow$=1, $\downarrow$=0 とすれば, 2 進数で [1110] と表すことができる。14=[1110] とすることで、状態を 2 進数表示すなわち bit で表現できる。交換項などは bit の入れ替えとして bit 演算を用いることで計算を効率的に行なうことができる。また、Hubbard 模型でも同様に表現が行える。Hubbard 模型の場合は up,down の電子をそれぞれ 0,1 で表現することで、2 進数で表せる。例えば, $|\uparrow\downarrow,\uparrow,\downarrow,0\rangle$ は [11,10,01,00] と表せる。

III.　HILBERT 空間の制限について

A.　2 次元サーチ法

これは、TITPACK で用いられている基本的な方法であり、Hilbert 空間を制限した場合にその逆引きを効率よく行なう方法である。まずは、この方法を解説する。例えば、totalS^z が一定の空間で考える場合は、前実空間配置の中から、指定した total S^z を持つものを抜き出して、一つ一つ

list に格納して行けばよい。問題はある実空間配置が制限した Hilbert 空間の中で、何番目の要素なのかを逆引きする場合である。一番、愚直な実装は全ての実空間要素に対して逆引きのリストもっておくことだがこれはあまりにもコストが大きすぎる。この逆引きを効率よくおこなうのが 2 次元サーチ法である [1,2]。

i	ib	ia	jb	ja	ja+jb
0 0 0 1 1 1	0 0 0	1 1 1	0	1	1
0 0 1 0 1 1	0 0 1	0 1 1	1	2	2
0 0 1 1 0 1	0 0 1	1 0 1	1	3	3
0 0 1 1 1 0	0 0 1	1 1 0	1	1	4
0 1 0 0 1 1	0 1 0	0 1 1	4	1	5
0 1 0 1 0 1	0 1 0	1 0 1	4	2	6
0 1 0 1 1 0	0 1 0	1 1 0	4	3	7

図 1: 2 次元サーチ法の図。i の bit を 2 つに分割して、分割した bit の情報から、何番目の要素かを再現する方法。

この方法の肝は 2 進表示した状態の bit を 2 つに分割することにある (これが 2 次元サーチ法の名前の由来)。図 1 に 6site, $S^z = 0$ の場合を示している。この場合、i を 2 つに分割した、上位 bit が ib, 下位 bit が ia である。jb, ja は以下のルールで逐次的に決めていく。

1. 最初は jb=0, ja=1
2. ib が変化しないなら、jb はそのままで ja を 1 つずつ増やしていく
3. ib が変化したときは、ja を 1 にして、jb はひとつ前の ja+jb の値にする

こうしておけば、ja+jb が i が何番目かの要素を示すようになる。それぞれの ib, ia に対して、jb, ja は一意に決めるので、list_jb[ib], list_ja[ia] という 2 つの配列 (配列の大きさは $2^(N/2)$ なので小さい) をもっておけば、i から ja+jb を生成することができるようになる。

2 次元サーチ法のアルゴリズムの擬似コードを **Algorithm** 1 に示す。例として N site の spin-1/2 の系を想定している。また,totalS^z =Nup としている (つまり,1 の bit の個数が Nup)。list_1 は要件

Algorithm 1 Two-dimensional search

```
jb← 0, ja← 0, icnt← 0, ib_old ← 0
for i=0 to 2^N-1 do
  if  number of bit in i = Nup then
    list_1[icnt] ← i
    ib ← get_ib(i)              //get ib from i
    ia ← get_ia(i)              //get ia from i
    if  ib = ib_old then
      ja ← ja+1
    else
      ib_old ← ib
      ja ← 1
      jb ← icnt-1
    end if
    list_ja[ia] ← ja
    list_jb[ib] ← jb
    icnt ← icnt+1
  end if
end for
```

をみたす (つまり所定の S^z を満たす) Hilbert 空間の元を格納する配列であり、get_ib,get_ia はそれぞれ i から上位 bit ib, 下位 bit ia をとり出す関数である。

B.　2 次元サーチ法のスレッド並列化について

Algorithm 2 Parallelization for two-dimensional search algorithm

```
jb← 0
for ib=0 to 2^{N/2}-1 do
  list_jb[ib] ← ib
  ib_Nup ← count_bit(ib)
  jb ← jb+Binomial(N/2,Nup-ib_Nup)
end for
for ib=0 to 2^{N/2}-1 do
  jb ← list_jb[ib]
  ib_Nup ← count_bit(ib)
  ja ← 1
  for ia=0 to 2^{N/2}-1 do
    ia_Nup = count_bit(ia)
    if ib_Nup+ia_Nup=Nup then
      list_1[ja+jb] ← ia+ib*ihfbit
      list_ja[ia] ← ja
      list_jb[ib] ← jb
      js ← ja+1
    end if
  end for
end for
```

さて、次は 2 次元サーチ法のアルゴリズムの並列化について述べる。逐次的な操作が入ってしまっているので、通常は並列化が不可能なように見えるが、ループを ib,ia に関するループに分割することで、並列計算が可能である。まず, 2 次元サーチのアルゴリズムから ib,ia のループを入れ子にすることが可能であす。ib で外側のループを回し、ia で内側のループを回せばよい。そして、ib に依る部分が jb だけだというところが味噌である。ib と jb は一対一に対応するので、ib に対応する jb がわかれば、ib に関するループは独立に計算でき、並列計算が行える。ib に対応する jb は ib に対応する ia の数を二項係数を計算することで、前もって知ることができる。この ib に関するループの部分は並列化不可能だがループ長が $2^{N/2}$ なので、この部分の計算時間は実質無視できる。

そのあとで、ib,ia の入れ子のループで外側の ib のループを openmp で thread 並列化すればよい。

擬似コードを **Algorithme** 2 に示す。ib,ia の 1 の bit の個数を数える関数 (count_bit) と二項係数を計算する関数 (Binomial) が必要だが、それ以外はもとの 2 次元サーチ法のアルゴリズムと同じである。また、ihfbit=$2^{N/2}$ で ia,ib からもとの i を復元するために必要な数である。

C.　同じ 1 の bit の個数をもつ次の bit を生成する方法

2 次元サーチ法の並列化のアルゴリズムをみると、並列化はされるが全体のループ長は依然として 2^N で計算コストのオーダーは変わっていないことがわかる。実はこの計算コストを縮めることも可能である。鍵となるアルゴリズムは「ある数と同じ 1 の bit の個数を持つ数のなかで次に大きい数を求める方法」である。これは「ハッカーの楽しみ」[3] という本に乗っている方法である [1] このアルゴリズムの擬似コードを **Algorithm** 5 にのせる。注意しなければならないのは x が 0 より大きくある必要がある必要があるということである。わずか、6 行で次の数が生成できる華麗なアルゴリズムである。

Algorithm 3 snoob: Get next larger bit

smallest ← x & (-x)

ripple ← x + smallest

ones ← x ^ ripple

ones ← (ones >> 2)/smallest

next_x ← ripple | ones

D.　さらなるスレッド並列化について

「ハッカーの楽しみ」の方法を用いて、さらなる効率化をしてみよう。まず、最初の 2 次元サーチ法のアルゴリズムそのものにこの方法をつかってもよいがそれだと並列化はできない。並列化バージョンに対して、ia のループの部分で「ハッカーの楽しみ」の方法を使うのことで並列化ができる。

Algorithm4 に擬似コードを示す。基本は 2 次元サーチ法の並列化版と同じで ia のループに関してのみ、snoob を用いている。snoob は 0 には使えないことに注意してほしい。これによって、計算コストは減少して、おちて N=36 といった巨大な系でもヒルベルト空間作成のコストはほぼ 0 になる。また、飽和磁場近傍のサイズ自体は大きいが、状態数が少ないところの計算も行えるようになっている。例えば、48 サイト 2up,46down といった条件のヒルベルト空間の作成はすぐに行える。

[1] この本の作者はこのアルゴリズムの紹介の前に、「いったい誰がそんなものを計算したがるのかという疑問が生じる」といっているが、実際に役に立つ例がここにあった！

Algorithm 4 Parallelization for 2D search algorithm II

```
jb← 0
for ib=0 to 2^{N/2}-1 do
  list_jb[ib] ← ib
  ib_Nup ← count_bit(ib)
  jb ← jb+Binomial(N/2,Nup-ib_Nup)
end for
for ib=0 to 2^{N/2}-1 do
  jb ← list_jb[ib]
  ib_Nup ← count_bit(ib)
  ja ← 1
  if ib_Nup ≤ Nup then
    ia = 2^{Nup−ib_Nup}-1
    if ia < ihfbit then
      list_1[ja+jb] ← ia+ib*ihfbit
      list_ja[ia] ← ja
      list_jb[ib] ← jb
      ja ← ja+1
      if ia ≠ 0 then
        ia ← snoob(ia)
        while ia < ihfbit do
          list_1[ja+jb] ← ia+ib*ihfbit
          list_ja[ia] ← ja
          list_jb[ib] ← jb
          ja ← ja+1
          ia ← snoob(ia)
        end while
      end if
    end if
  end if
end for
```

IV. FERMION サインの数え方について

ハバード模型など遍歴電子系のフェルミオンサインについて説明する。つねに粒子がいるスピン系とは違い、遍歴電子系の場合は電子がいないサイトがあるので、ホッピング項などを作用させるときにフェルミオンの交換関係から符号が生じる。例えば、以下のような場合である。$c^{\dagger}_{2\uparrow}c_{0\uparrow}|0,0,\downarrow,\uparrow\rangle = |0,\uparrow,\downarrow,0\rangle$ この場合、ホッピングする間の 1 の bit の偶奇を計算する必要がある。

ここでも「ハッカーの楽しみ」[3] にのっている方法が役にたつ。与えらた数の 1 の bit のパリティをを求めるアルゴリズムを用いて、元の bit(orbbit) から ±1 の sgn を求めている。もとの orgbit は 64bit の整数を仮定している。これによって計算が高速化できる。

Algorithm 5 parity counting

```
bit ← orgbit ˆ (orgbit>>1)
bit ← bitˆ (bit>>2)
bit ← bitˆ (bit>>4)
bit ← bitˆ (bit>>8)
bit ← bitˆ (bit>>16)
bit ← bitˆ (bit>>32)
sgn ← 1-2*(bit & 1); //sgn=±1
```


1 http://www.stat.phys.titech.ac.jp/ nishimori/titpack2_new/index-e.html.

2 H. Q. Lin, Phys. Rev. B **42**, 6561 (1990).

3 S. H. Warren, Hacker's Delight (2nd Edition) (Addison-Wesley, 2012), ISBN 0321842685.