汎用プラスチクスにおけるエンタルピー緩和のモデル化と並列処理プログラムの作成

坂本　宜紀*　　田中　穣*

Modelling of the Enthalpy Relaxation in Conventional Plastics with the Parallel Computing Program

Noriki SAKAMOTO* and Yutaka TANAKA*

(Received July 8, 2015)

As a background of this study, there is an analysis of the enthalpy relaxation in the high polymer thermal history. We made a calculation program to predict enthalpy relaxation. The prediction of the enthalpy relaxation is important from a point choosing materials in the manufacturing. However, the model needed long-time for calculation to predict enthalpy relaxation. Therefore we tried that we incorporated parallel computing in a program to shorten calculation time.

Key Words : Enthalpy Relaxation, TNM framework, Model calculation, OpenMP

1. 緒　言

現代社会において多くの製品に利用されている汎用プラスチクスがある．これらプラスチクス製品は，経時変化に伴う劣化スピードが金属無機材料や天然素材に比べ速い事が明らかとなっている．これは，熱や力の経時変化により材料の物性が安定状態に近づく事が原因とされている．この現象の一つとしてエンタルピー緩和がある．エンタルピー緩和は，経時変化に伴いエンタルピーが安定な状態に緩和する事である．つまり、エンタルピー緩和が起こると材料は脆くなるという事である．製品化される汎用プラスチクスは，エンタルピー緩和を考慮した上で選択されている．我々は，汎用プラスチクスの中のポリスチレンについて，エンタルピー緩和に関する解析を行った．エンタルピー緩和の研究方法として，DSC 測定装置を用いた熱処理実験[1]と Tool-Narayanaswamy-Moynihan フレームワーク[2]-[4]を用いた計算による比熱(c_p)のシミュレーションの2つがある．今研究では計算シミュレーションの方をテーマとしている．

現在，我々の研究室ではエンタルピー緩和を予測する(c_pを計算する)システムとして，s047.f90 という計算プログラム[5]を使用している．s047 は，アダム・ギブスの理論[6]-[7]に基づいており S_c(配位エントロピー)より c_p を算出できる(S_c モデル)．DSC 実験データ[1]を再現する事ができ，A(高温極限の緩和時間/min)，B(エネルギー定数/Jg)，T_2(Gibbs-Dimardio 温度/℃)[6]というパラメータの値を算出する事が出来る．パラメータの算出の概要は，A, B, T_2 を 3 重の do ループにする事で，最も DSC 実験データを再現出来ていた際の値を探す方法をとっている．3 重 do ループを組むことで，全周回数は 1000 万以上となり，1 日以上の計算時間を必要としていた．そこで，計算時間の短縮を目的とし，プログラムの並列処理を試みた。

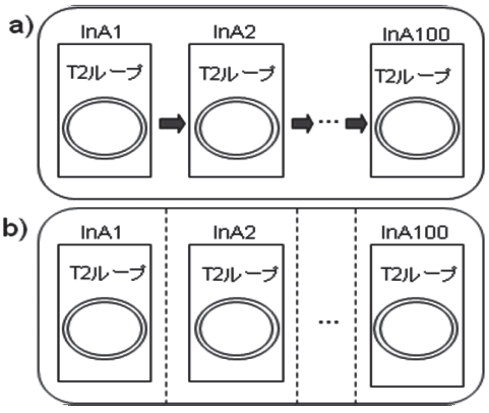

Fig. 1　並列化のイメージ図.
a) 従来のプログラムのイメージであり，直列つなぎのように，lnA1 が終わり lnA2 へ進む流れ.
b) 並列処理のイメージであり，並列つなぎのように，lnA1～lnA100 が全て同時進行する流れ.

*大学院工学研究科材料開発工学専攻
*Materials Science and Engineering Course, Graduate School of Engineering

プログラムの並列処理とは, do ループを並列処理(同時進行)することで格段な時間短縮が望めるというシステムである. 我々は, このシステムを s047.f90(S_c Model サーチルーチン型)のプログラムに組み込む事を試みた. Fig.1 にイメージ図を示した. 図に示したように, 最も外側に存在する lnA のループを並列化することを試みた. 初めは、並列処理を学ぶために参考書(Nag の OpenMP 入門)[8]をもとに理解を試みた.

2. 解析と実験

2.1 並列段階 I (sections 指示構文の解析)

「Nag の OpenMP 入門」の 6 章 sections 指示構文を理解し, 実際に動かす事を目的とした. 演習課題として提示されているコードを kadaiSections.f90 とした. Fig. 2 を参照.

このソースコードは, N 個の乱数(a(N)と b(N))を小さい順に並べ替えるプログラムである. sort(a)では a (N)を並べ変え, sort(b)では b(N)を並べ変えている. 2 つの sort は並列に進行している. つまり, 通常は a(N)の並べ替えの後に b(N)の並べ替えを行うところ, sections 指示構文を利用する事で同時に実行することが出来るのである.

Fig.2 における 8〜15 行目のコードが最も重要となっている. まず, 8 行目と 15 行目の parallel 指示構文で並列を命令できる領域を作る. 次に, 9 行目と 14 行目で sections 指示構文の領域を作る. parallel 領域の中でしか sections 指示構文は使用できないためである. そして, 9 行目と 10 行目や 11 行目と 12 行目で sections 指示構文によって並列化を行う. !$となっているが, 実際は無視されずに実行される. 本文は sort(a)と sort(b)を call するための 2 並列である. 呼び出された sort(a)と sort(b)はそれぞれ 19 行目の subroutine sort(x)に対応している. subroutine sort(a)と subroutine sort(b)が並列に進行するということである. subroutine sort(a)では x(:)=a(:), subroutine sort(b)では x(:)=b(:)となっている. また, subroutine が始まる

```
1   program sections
2     implicit none
3     integer,parameter :: N=10000   !変数宣言 3〜5 行目
4     real, allocatable :: a(:), b(:)
5     allocate(a(N), b(N))
6     call random_number(a)      !0 以上 1 未満の乱数を a(1)〜a(N)に設定
7     call random_number(b)      !                          b(1)〜b(N)に設定
8   !$omp parallel             !parallel 指示構文. 並列領域の開始.
9   !$omp sections             !sections 指示構文.
10  !$omp section              !別のスレッドで実行
11    call sort(a)             !サブルーチンの呼び出し 19 行目
12  !$omp section              !別のスレッドで実行
13    call sort(b)             !サブルーチンの呼び出し 19 行目
14  !$omp end sections         !sections 指示構文の終わり.
15  !$omp end parallel         !並列領域の終了.
16    print '("first 4 number in a : ",4f10.7)', a(1:4)      !サブルーチン終了後の a(1)〜a(4)をプリント.
17    print '("first 4 number in b : ",4f10.7)', b(1:4)      !サブルーチン終了後の b(1)〜b(4)をプリント.
18  contains
19    subroutine sort(x)         !サブルーチンの開始. x を a や b に置き換えて読む.
20      real x(:), tmp           !変数宣言 20〜21 行目
21      integer i, j
22      do i = 1, N-1            !22 と 30 で i の do ループ.
23        do j = i + 1, N        !23 と 29 で j の do ループ. i と j の 2 重ループ.
24          if (x(i) > x(j))then     !24〜27 では, x(1)〜x(N-1)までで, 小さいものから x(1), x(2)とする.
25            tmp = x(i)             !並べ替え処理
26            x(i) = x(j)
27            x(j) = tmp
28          end if
29        end do
30      end do
31    end subroutine sort    !サブルーチンの終了.
32  end program sections     !プログラムの終了.
```

Fig. 2　kadaiSections.f90 のソースコード. Sections 指示構文を利用した並列処理プログラム.

前には必ず，contains を付けなければならない．subroutine は 19 行目から 31 行目であり，その中できる事が分かる．subroutine 終了後は 14 行目に戻り，15, 16, 17, 32 行目と進行し，プログラムは終了する．以上が kaidaiSections.f90 の内容と流れである．

今後の展開としては，kadaiSections.f90 を基盤として，s047.f90 で用いている c_p calculation や多重 do ループによるパラメータ値の算出を追加していき，並列処理プログラムを完成させる事を目指す．

2.2 並列段階Ⅱ(kadai sections の応用)

kadaiSections.f90 に基本的なコードを用いる事が出来るか確認する事を目的とした．本研究では，kadaiSections.f90 を少しずつ改良していく際のソースコードを Sections 番号.f90 の形で表記する．

第一の過程として Sections01.f90 では，サブルーチンの中で input ファイルを開き，中身を読み込み，ファイルを閉じる事を試みた．input ファイルからは，k という定数の値を読み込んでいる．このプログラムは，sort(a)と sort(b)の 2 並列であり，1 つの input ファイルを 2 つのスレッドで共有することになっている．次に Sections02.f90 として，サブルーチンに対応させる変数の変更を行った．作成したソースコードの詳細を Fig. 3 で示す．サブルーチン内の割り付け変数

```
1   program sections
2     implicit none
3     integer,parameter :: N=10000
4     real(8), save::a, b
5   a=10
6   b=100
7   !$omp parallel
8   !$omp sections
9   !$omp section
10    call sort(a)
11  !$omp section
12    call sort(b)
13  !$omp end sections
14  !$omp end parallel
15  contains
16    subroutine sort(x)
17      real(8) x
18      integer k
19  open(10, file='input.txt')
20  read(10,*) k
21  close(10)
22  print *, k*x
23    end subroutine sort
24  end program sections
```

Fig. 3　Sections02.f90 のソースコード
並列処理を行う際に，input ファイルの挿入が可能である事が分かった．

x(:)に対応している a(:)と b(:)について，それぞれ変数 x と a と b に変更した(4 行目と 17 行目)．これは，割り付け変数で対応させるより，変数で対応させた方が簡単であると考えたからである．実際にソースコードでは，a と b は定数として用いている(5〜6 行目)．

今後の展開として，サブルーチン内に c_p calculation を入れる事や do ループにより，マルチパラメータのループを導入する事、さらに最適値の算出を行う事が挙げられる．

2.3 並列段階Ⅲ(c_p calculation の挿入)

kadaiSections.f90 に対して s047.f90 で用いている c_p calculation のソースコードを徐々に追加していき，cp を計算できるようにする事を目的とした．この段階では，並列をあまり意識しておらず，sort(a)と sort(b)の 2 並列で行っている．

まず，Sections03.f90 としてサブルーチンの中に s047.f90 の温度条件パラメータの入力という部分を追加した．Fig. 4 に内容を示す．新しく増えた変数の宣言はサブルーチン内で行った．また，Input ファイルの名前を Input_capa_cal.d に変更完了した．Input ファイルから温度条件パラメータを読み込み，それをもとに Y1〜Y7 までの値を計算する事をしている．詳細を Fig. 4 に示す．

```
open(9, file = 'Input_capa_cal.d')
   read(9,*) T0, Ta, Tb, Tc, Tg, Q1, Q2, Q3, beta, B
close(9)
T0=T0+273.15 ; Ta=Ta+273.15 ; Tb=Tb+273.15 ;
Tc=Tc+273.15 ; Tg=Tg+273.15
a1=-0.000806 ; b1=0.461772
a01=0.00774 ; b01=-1.08833 ; S1=4d0 ; k=0d0
dT=0.5
Tdet=313.15d0 !40℃を入力
Y1=(T0-(Tg+10))/dT ; Y2=Y1+((Tg+10)-Ta)/dT ;
Y3=Y2+S1 ; Y4=Y3+(Ta-Tb)/dT
Y5=Y4 + ((Tg-Tb)/dT) - 5/dT ;
Ydet=Y4 + (Tdet - tb)/dt ; Y7=Y4+(Tc-Tb)/dT
dat=-dT
```

Fig. 4　温度条件パラメータの入力
c_p calculation を行う前に，必要となるパラメータの値を求めている部分である．

次に，Sections04.f90 として s047.f90 で重要となるパラメータ ln$A$, T_2, G_0 を追加した．ln$A$=x, T_2=25, G_0=0.08 として直接入力した．ここで ln$A$ に x の値を用いることが，並列処理を行うにあたって大切なポイントとなる．x を ln$A$ として活用すると，ln$A$=a または ln$A$=b ということになる．本研究ではこの

Sections04.f90 の章で, ./a.out(プログラムの実行)を行う際に実行出来る時と出来ない時がある事態が発生した. しかし, 原因の糸口が掴めなかったため現時点では保留にして, 先に進むことにした.

Sections05.f90 では, s047.f90 の End of The Second の部分までを追加した. Fig.5 に追加したコードの詳細を示す. この部分では, 番号 n に対して温度を与える事を行っている. n は時間に相当する変数であり, n が 1 つ増える毎に温度が 0.5K ずつ変化する. n=0 のとき温度 T=433.15K(160℃)である. Y1～Y7 も番号であり, n に代わって使用することが可能である. また, この章では c_p calculation の第 1 冷却ステップのコードの追加も行っている. 第 1 冷却ステップとは, Y1～Y2 の区間であり, アニール前までの過程を意味する. このステップにおける, 配位エントロピーS_cや緩和時間τの値を計算している. 温度 T, 番号 n, Y1～Y7, ステップの関係を温度プログラムの形で Fig. 6 に示す. 図における T_0 は実験開始温度, T_A はアニール温度, T_b は昇温開始温度, T_c は昇温終了温度, Q_1 は第 1 冷却速度, Q_2 は第 2 冷却速度, Q_3 は昇温速度, t_A はアニール時間を意味する.

```
Do i=1, Y7
        T_N(i)=T0+i*dat ; G(i)=0
        if (Y1 < i) then
        G(i) =(Tg + 10 - T_N(i))*G0/15 ; end if
        if ((Y1+15/dT) < i) then
        G(i) = G0 ; end if
        if (Y2 < i) then
        dat=0 ; T_N(i)=Ta ; end if
        if (Y3 < i) then
        dat=-dT; T_N(i)=T0+(i-S1)*dat ; end if
        if (Y4 < i) then
        dat=dT ; T_N(i)=Tb+(i-Y4)*dat ; end if
        if ((Y4+((Tg-5) - Tb)/dT) < i) then
        G(i)=(Tg + 10 - T_N(i))*G0/15 ; end if
        if ( (Y4+((Tg+10)-Tb)/dT) < i) then
                G(i)=0 ; endif
  J(i) = a1*(T_N(i) - T2) + b1*log(T_N(i)/T2) –
        G(i)*log(T_N(i)/(Tg+10))
  L(i) = a1*dat + (b1 - G(i))*log(T_N(i)/(T_N(i) - dat))
enddo
        p_MM=0; p_ML=0; dat=-dT
        Do i=Y1, Y2
          Sc(i) = J(i) - p_MM
          tau(i) = exp(lnA + B/(T_N(i) * Sc(i)))
          dttau(i+1) = dat/(Q1*tau(i))
          D = 0; p_MM = 0; p_ML = 0
            Do m = 1, i
            D = D + dttau(i-m+2)
            p_ML = L(i-m+2)*( exp(-1*D**(beta)) )
            p_MM = p_MM + p_ML
          EndDo
        EndDo
!======================= End of The Second.
```

Fig. 5　温度 T の設定と第 1 冷却ステップのコード
温度 T を n という番号によって与えた. そして, c_p calculation の初めのステップを追加した.

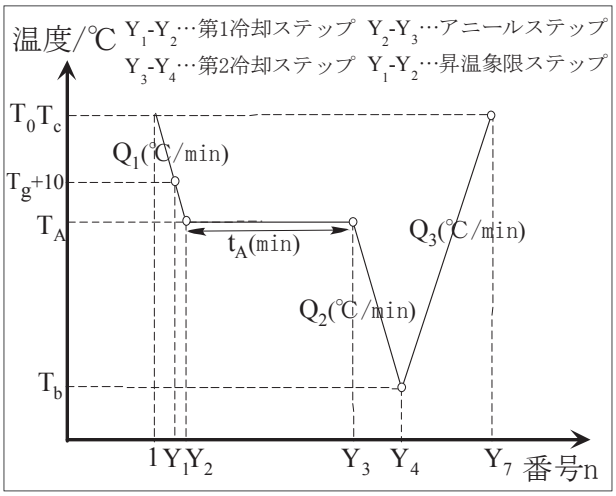

Fig. 6　s047.f90 に用いた温度プログラム
Y1～Y7 と対応するステップを示した. この温度プログラムは実際に DSC 実験で用いた形を採用している.

Sections06.f90 では, s047.f90 の End of The Third (Anneal)の部分までを追加した. この章では, アニールステップにおける, S_c やτの計算をしている. 詳細を Fig. 7 に示す.

```
Do i=Y2+1, Y3
        k = k + 1
        if (i == Y2+1) then
        tk = (te)**(k/S1)
        else
        tk = (te)**(k/S1) - (te)**((k-1)/S1); end if
          Sc(i) = J(i) - p_MM
          tau(i) = exp(lnA + B/(Ta * Sc(i)))
          dttau(i+1) = tk /tau(i)
          D = 0; p_MM = 0; p_ML = 0
           Do m = 1, i
             D = D + dttau(i-m+2)
             p_ML = L(i-m+2)*( exp(-1*D**(beta)) )
             p_MM = p_MM + p_ML
           EndDo
  EndDo
!======================End of The Third(Anneal).
```

Fig. 7　アニールステップのコード
アニールステップにおける計算だけ, 他のステップとは異なる計算をしている事に注意する.

Sections07.f90 では, s047.f90 の End of The Forth and Fifth の部分まで追加した. 詳細を Fig. 8 に示す. 第 2 冷却ステップと昇温ステップの 2 つを同時に追加している. Y3～Y4 と Y4～Y7 に分けるわけではなく, Y3～Y7 としてループの中で if 文を使って場合分けしている. この章で, すべてのステップにおける S_c やτの計算が完了となる. この後は, S_c やτをもとに c_p を計算していく流れである.

```
   Do i=Y3+1, Y7-1
            if (Y3<i) then
            dat= -dT; Qs=Q2 ; end if
            if (Y4<i) then
            dat= dT; Qs=Q3 ; end if
            Sc(i) = J(i) - p_MM
            tau(i) = exp(lnA + B/(T_N(i) * Sc(i)))
            dttau(i+1) = dat/(Qs*tau(i))
             D = 0; p_MM = 0; p_ML = 0
        Do m = 1, i
             D = D + dttau(i-m+2)
             p_ML = L(i-m+2)*( exp(-1*D**(beta)) )
             p_MM = p_MM + p_ML
        EndDo
   EndDo
!=================End of The Forth and Fifth..
```

Fig. 8　第 2 冷却ステップと昇温ステップのコード
2 つのステップを同時にループするため,
パラメータの場合分け(if 文)が重要となる.

Sections08.f90 では, s047.f90 の H^(eq) for The First – The Fifth の部分を追加した. 詳細を Fig. 9 に示す. 1～Y7 に対応したすべての温度に対して, 配位エンタルピーH_c を算出している. if 文によって場合分けを行うことで 1 つのループで各ステップを再現している. ソースコードの本文では, dttau と F が領域外へアクセスしない様に Do i = Y1, Y7-1 とした. 領域外のアクセスとは, 実際には存在しないものを呼び出す事を意味する. つまり, dttau(Y7)や F(Y7)が存在しないのに do ループを Y7 まで組むとエラーが起こるという事である. この Sections08.f90 の章で H_c を計算したのは, c_p を算出する際に, S_c からではなく H_c を用いるためである.

Sections09.90 では, s047.f90 の cp calculation の部分を追加した. この章で c_p を計算することが可能になる. 追加したコードの詳細は Fig. 10 に示す. 本文では, cp_2 として c_p(比熱)を計算しているが, 都合上 cp_2(Y7-1)など 1 点しか出力することが出来ない. Sections09.f90 のプログラムで並列段階Ⅱが完成した. 実際に内容を示す. パラメータの値を入力する事で, $c_p(T)$を計算するプログラムになった. sort(a)と sort(b)といった 2 つのサブルーチンが同時に進行し, ln$A$=a と ln$A$=b の異なる c_p(T)を計算している. sort(a)と sort(b)の違いは, lnA だけで他は全く同じである. $c_p(T)$は計算出来ているが, カーブを作成する事はできない. ln$A$(または x), T_2, G_0, N は定数を用いており, ループはしていない. 以上が, 並列段階Ⅱにおける状況である.

```
   dat = -dT
      Do i=1, Y7
            if (Y2 < i) then
            dat=0; end if
            if (Y3 < i) then
            dat=-dT; end if
            if (Y4 < i) then
            dat=dT ; end if
  E(i)= a1*(T_N(i)**2 - T2**2)/2 +
        b1*(T_N(i) - T2) - G(i)*(T_N(i) - (Tg + 10))
  F(i)= a1*(T_N(i)**2 - (T_N(i) - dat)**2)/2 +
       (b1 - G(i))*dat
      EndDo
      dat = -dT; p_MK = 0; p_MN = 0
         Do i = Y1, Y7-1
            if (Y2 < i) then
            dat=0; end if
            if (Y3 < i) then
            dat=-dT; end if
            if (Y4 < i) then
            dat=dT ; end if
            Hc(i)= E(i) - p_MN    !This is Hc(t).
              D = 0; p_MK = 0; p_MN = 0
              Do m = 1, i
                 D = D + dttau(i-m+2)
                 p_MK = F(i-m+2)*( exp(-1*D**(beta)) )
                 p_MN = p_MN + p_MK
              EndDo
         Enddo
```

Fig. 9　すべてのステップにおける H_c の算出.

```
   Do i=Y4, Y7
      cp_1(i) = a01*T_N(i) + b01 + (Hc(i) - Hc(i-1))/dT
   EndDo
   derutA = cp_1(Y5) - cp_1(Y5 + 1)
   if (G0 == 0) then
   derutA = 0 ; end if
      Do i= Y4, Y7
         if (Ydet < i) then
shif(i) =(derutA/(2*(Tg-5-Tdet)))*(T_N(i) - Tdet); end if
         if (Y5 <i) then
shif(i)= derutA*T_N(i)/(2*14.5) -(derutA*(Tg+10)/(2*14.5))
         end if
         if ((Y5 + 15/dT) < i) then
         shif(i)=0d0 ; end if
         cp_2(i) = cp_1(i) - shif(i)
      EndDo
```

Fig. 10　c_p の算出.

次の段階で行うべき項目をまとめる. 1 つ目は, マルチパラメータのループを組み込む事である. そのためには, T_2, G_0 のループの上限と下限値を読み込む必要がある. N のループを組み込む際には, 熱処理時間 tR(N)や DSC 実験データが必要となってくる. tR(1)=0, tR(2)=28, tR(3)=82, tR(4)=290, tR(5)=1012, tR(6)=1910 を与えるようにする. 2 つ目は, lnA のループをサブルーチン毎に独立させることである. 現段階では, a と b の数値が lnA となっているため, lnA のループに利用できると考えた. lnA を独立させる案を Fig. 11 に概要として示す.

サブルーチン外

```
a=-50
b=-100
   :
call sort(a)
   :
call sort(b)
   :
```

サブルーチン sort(a)

```
x=-50
do z = 0, 50/1    !50=間隔, /1=キザミ
lnA = x-z*1       !lnA=-50, -51,,,,-99, -100
end do
```

サブルーチン sort(b)

```
x=-100
do z = 0, 50/1    !50=間隔, /1=キザミ
lnA = x-z*1       !lnA=-100, -101,,,-150
end do
```

Fig. 11　並列 do ループの仕組み
ln$A$ ループを各サブルーチンで独立させる方法.

この考えをもとに, 次の段階では並列を意識して do ループを組み込む事を試みた.

2.4 並列段階Ⅳ(do ループと並列処理)

サブルーチン内でマルチパラメータのループを組み, lnA のループを並列させる事を試みた. T_2 や G_0 や N に関しては s047.f90 と同じようにループを組み, lnA に関しては Fig. 10 を参考に作成してみた.

Secstions10.f90 では, T2 ループと input_T2_lnA.d の追加を行った. input_T2_lnA.d より T2_dai(上限値)と T2_sho(下限値)を読み込んでいる. input_T2_lnA.d はサブルーチン内で呼び出している.

Sections11.f90 では, G0 ループと N ループと input_yomikomi.d の追加を行った. input_T2_lnA.d 内に G0_dai と G0_sho の追加を行い, T2 と同じように読み込んだ. input_yomikomi.d では tR(N)だけ読み込んでおり, サブルーチンの外で呼び出している. この章では, exp_data はまだ使用していない.

Sections12.f90 では, lnA のループを組み, 並列処理させる事を試みた. a と b の数値をそのまま用いてサブルーチンの中の lnA のループに利用するため Fig. 10 をコードに追加した. 本文では、a=-10, b=-20 としてサブルーチン内で z=0～9 のループを組み lnA=a - z として lnA をループさせている. Sections10～12 で追加した項目を Fig. 12 に示す. 各コードにおいて, 新しく使用した変数を宣言する事を忘れないように注意する.

```
   open(10,file='input_yomikomi.d')
      do i=1,N
         read(10,*) tR(i)
      enddo
   close(10)
      :
!サブルーチン開始
      :
   open(2,file='Input_T2_lnA.d')
   read(2,*) T2_sho, T2_dai, G0_sho, G0_dai
   close(2)
do z = 0, 9
   do y = 0, (T2_dai - T2_sho)
      do w = 0, nint((G0_dai - G0_sho)*10)
         do s = 1, N
   open(9, file = 'Input_capa_cal.d')
    read(9,*) T0, Ta, Tb, Tc, Tg, Q1, Q2, Q3, beta, B
   close(9)
      :
   lnA = x - z
   T2 = T2_sho+273.15+y
   G0 = G0_sho+w*0.1
   te=tR(s)
      :
         end do !N(s)
      end do !G0(w)
   end do !T2(y)
end do !lnA(z)
```

Fig. 12　並列段階Ⅳで追加した項目
マルチパラメータの do ループを組むことに成功した. lnA を各サブルーチンで独立させる事にも成功した.

現時点で, $c_p(T)$の計算やマルチパラメータのループや lnA の並列が完了した. 残ることは, 最適値を算出することである. そのためには, exp_data の読み込み, ラベリング点(La_exp～Lh_exp, La_cal～Lh_cal)の決定, Sum(最小二乗法による偏差)の計算, 最適値の算出, などが挙げられる. 最適値とは, 最も実験値を再現した時のマルチパラメータの値である. 実験データの再現率を計算するために, ラベリング点や Sum の計算などを行っている. なお, この最適値の算出が完了すると, s047.f90 の並列化は完成すると考える.

2.5 並列段階Ⅴ(最適値の算出)

プログラムで最適値を算出できるようにするため,

最小二乗法を追加する．サブルーチン毎に異なるループを行い，それぞれで最適値を算出する．最終的な最適値は，各サブルーチンから出力された値から選ぶ形になるだろう．

Sections13.f90 では，Input_yomikomi.d から file 名とラベリング温度(P_a～P_h)を読み込めるようにした．file 名を読み込んだため，exp_data をプログラムに取り込む事に成功した．そのときの mins は 600 と設定した．mins とは読み込むデータの行数を意味する．また，ラベリング温度を読み込んだため，La_exp～Lh_exp といった実験値のラベリング点の決定も行うことに成功した．これらは全てサブルーチンの外で行っている．N=6 とする事で，6 個の exp_data を使用した．exp_data の読み込み方法とラベリング点の決定方法を Fig. 13 に示す．

```
mins=600
do q = 1, N
 open(30+q,file=name(q))
 do w = 1,mins
  read(30+q,*) Tt(q,w),cp_exp(q,w)
 end do
 close(30+q)
     do w = 1,mins
        if (Tt(q,w)== P_a(q)) then
           La_exp(q) = cp_exp(q,w)    ; end if
        if (Tt(q,w)== P_b(q)) then
           Lb_exp(q) = cp_exp(q,w)    ; end if
        if (Tt(q,w)== P_c(q)) then
           Lc_exp(q) = cp_exp(q,w)    ; end if
        if (Tt(q,w)== P_d(q)) then
           Ld_exp(q) = cp_exp(q,w)    ; end if
        if (Tt(q,w)== P_e(q)) then
           Le_exp(q) = cp_exp(q,w)    ; end if
        if (Tt(q,w)== P_f(q)) then
           Lf_exp(q) = cp_exp(q,w)    ; end if
        if (Tt(q,w)== P_g(q)) then
           Lg_exp(q) = cp_exp(q,w)    ; end if
        if (Tt(q,w)== P_h(q)) then
           Lh_exp(q) = cp_exp(q,w)    ; end if
     end do
end do
```

Fig. 13　exp_data の読み込みとラベリング点の決定
mins=600 と設定する事で，exp_data 内の 600 行目まで読み込むことが可能である．実験値のラベリング点は，if 文を用いて，ラベリング温度と同じ温度の c_p を決定している．

Sections14.f90 では，La_cal～Lh_cal という計算値のラベリング点の決定を行った．さらに、(La_exp～Lh_exp)や(La_cal～Lh_cal)の値をもとに，Sum の計算と Sum_total の計算も行った．Sum は，ある温度(ラベリング温度)における実験値と計算値の差の二乗を足し合わせたものである．最小二乗法と呼ばれる方法である．また Sum_tatol は Sum を足したものである．本紙では N=6 であるため，Sum_total=Sum(1)+ Sum(2)+ Sum(3)+ Sum(4)+ Sum(5)+ Sum(6)となっている．これにより，パラメータの組み合わせ毎の標準偏差が得られ，偏差の値が小さいほど実験値を再現出来ている事が分かる．

Sections15.f90 では，S_sum の決定を行い，最適値の算出を行った．S_sum とは，Sum_total(偏差)の中で最も小さいもののである．sort(a)と sort(b)の 2 並列であり，それぞれで最適値を算出しているが，この 2 つからさらに絞り込む点は目視で行っている．実質上この Sections15 において完成系と考えている．s047.f90 とは異なる点は，時間換算表示のシステムが組み込まれていないだけだと考える．Sum の計算と sum_total, S_sum の決定のコードを Fig. 14 に示す．

```
        :
Sum(s)=(La_exp(s)-La_cal)**2+(Lb_exp(s)-Lb_cal)**2&
      &+(Lc_exp(s)-Lc_cal)**2+(Ld_exp(s)-Ld_cal)**2&
      &+(Le_exp(s)-Le_cal)**2+(Lf_exp(s)-Lf_cal)**2&
      &+(Lg_exp(s)-Lg_cal)**2+(Lh_exp(s)-Lh_cal)**2
Sum_total = Sum_total + Sum(s)
         end do !N
   if (S_Sum > Sum_total) then
      S_sum = Sum_total
      Sum_A = lnA
      Sum_T2 = T2
      Sum_G0 = G0 ; endif
```

Fig. 14　Sum の計算と sum_total, S_sum の決定
Sum は実験値と計算値の差の二乗を足したものであり，Sum_total は N のループを利用して N 個足したものである．S_sum は N ループの外で行うことに気をつける．

2.6 並列段階VI(並列処理の増加)

並列処理数を 2 並列から 4 並列へ増大をさせた．

Sections16.f90 では，サブルーチンとして呼び出す sort の数を 2 つから 4 つへ増やした．正確には，sort(c3)と sort(c4)を追加した．しかし，4 並列にすると a.out を実行出来なくなった．これは，2 並列(Sections04.f90)の時に現われていたエラーである．何度か実行するとエラーが出ない時があったので，ここまで無視して来た．この原因としては，Input_capa_cal.d への集中が考えられた．4 つのスレッドが同時進行しているため，1 つが利用していると他が利用できない事が理由であった．

そのため Sections17.f90 では，サブルーチン内で呼び出している input ファイルである Input_capa_cal.d

と Input_T2_lnA.d への集中をなくすため, 読み込む場所をサブルーチンの外で行うように変更した. その結果, 4 つのサブルーチンを並列処理する事に成功した. 4 並列の際の指示構文を Fig. 15 に示す.

```
!$omp parallel
!$omp sections
!$omp section
   call sort(c1)
!$omp section
   call sort(c2)
!$omp section
   call sort(c3)
!$omp section
   call sort(c4)
!$omp end sections
!$omp end parallel
```

Fig. 15　4 並列を行う際の section 指示構文.

Sections17.f90 においてプログラムが完成したかを確認するため, 原本となった s047.f90 のプログラムとサーチ結果を比較した. s047.f90 と Sections17.f90 は同じ計算を使用しているのでサーチ結果も同じになるはずだと考えた. そこで, 入力した条件パラメータ(T_0, T_a, T_b, T_c, T_g, Q_1, Q_2, Q_3, β, B), 係数(a_{01}, b_{01}, a_1, b_1), exp_data, サーチ範囲(ln$A$, T_2, G_0), キザミ幅などすべての条件を統一し, それぞれサーチを行った.

設定したサーチ範囲と入力条件を以下に示す. -200 < lnA < 0 ; キザミ 2, -20 < T2 < 90 ; キザミ 2, 0 < G0 < 0.12 ; キザミ 0.03, exp_data 数 N=6, T_0=160, T_a=94, T_b=20, T_c=160, T_g=102, Q_1=-12, Q_2=-9.5, Q_3=5, β=0.330, B=1000 を用いた.

Secctions17.f90 では, ln$A$ のループを並列処理するため, 各 sort の担当範囲を設定した(Fig. 16 参照).

sort(c1)=-50<lnA<0	,	sort(c2)=-100<lnA<-50
sort(c3)=-150<lnA<-100	,	sort(c4)=-200<lnA<-150

Fig. 16　並列処理の際の lnA の分割.

それぞれ最適値(最も実験値を再現した値)を算出した後, 4 つの最適値から更なる最適値を目視で選ぶシステムである.

サーチした結果を Fig. 17 に示す. Sections17.f90 の最適値は, うまく算出されなかった. これを改善するには, s047.f90 のソースコードと Sections17 のソースコードを照らし合わせて異なる点を改善するしかないと考えた. 最適値が選ばれない原因は, Input_capa_cal.d の読み込みとパラメータの決定の場所(サブルーチンの外に移動した箇所)にあった. これらはループの内側になければならないことが分かったため, 所定の位置に戻すことにした. しかし, サブルーチンの中では Input ファイルが使えない(各サブルーチンが 1 つのファイに集中するため)ので Input_capa_cal.d で読み込むパラメータ値をソースコード上で直接入力する方法をとった.

```
s047.f90
        …S_sum = 0.1211, lnA = -54.0, T2 = 18.0, G0 =0
Sections17.f90
Sort(c1)…S_sum = 3.1042, lnA = 0, T2 = -20.0, G0=0
Sort(c2)…S_sum = 0.9110, lnA = -50.0, T2 = -20.0, G0=0
Sort(c3)…S_sum = 0.9110, lnA = -100.0, T2 = -20.0, G0=0
Sort(c4)…S_sum = 0.9110, lnA = -150.0, T2 = -20.0, G0=0
```

Fig. 17　s047.f90 と Sections17.f90 のサーチ結果.

2.7 バッチ形式における並列処理(スクリプト)

並列化プログラムをバッチ形式で使用する際は, スレッド数やコア数に注意する必要がある. それらはすべてスクリプトで宣言する事が出来る. スクリプトの中身(Queuesys.sh)を Fig. 18 に示す.

```
#!/bin/csh
#============= LSF Options =============
#QSUB -q   gr10037b
#QSUB -ug gr10037
#QSUB -J   s060
#QSUB -W   200:00
#QSUB -A p=1:t=16:c=16:m=1920M
#============= Shell Script ============
set -x
time ./a.out >& ProgressRepSEPTEMVER19.dat &
```

Fig. 18　バッチ形式におけるスクリプトの内容.

ここで注目するのは「#QSUB -A p=1 : t=16 : c=16 : m=1920M」の行についてである. p, t, c, m の意味を以下に示す. p･･･プロセス(process)の数. t･･･プロセスあたりのスレッド(threads)の数. c･･･プロセスあたりのコア(cores)の数. m･･･実行するにあたる必要メモリー(memory). 並列化を実行するときに必要となるスレッド数以上の数値を t に入力しておかなければならない. t=1 の際には, 並列処理はされず c1→c2→c3→c4 と順番に処理されてしまう. つまり, 並列させる数以上のスレッドが必要だということである.

Sections18.f90 において s047.f90 のサーチ結果と比較した. 設定したサーチ範囲と入力条件は先ほどの並列段階Ⅵの時と同じである.

Sections18.f90 の ln*A* のループの設定も Fig. 16 の値を使用した. 目標となる s047.f90 におけるサーチ結果は, S_sum = 0.1211, lnA = -54.0, T2 = 18.0, G0 =0 であった.

```
・会話型
Sort(c1)･･･S_sum = 0.2274, lnA = -50, T2 = 12.0, G0=0
Sort(c2)･･･S_sum = 0.1211, lnA = -54.0, T2 = 18.0, G0=0
Sort(c3)･･･S_sum = 0.5476, lnA = -102.0, T2 = 54.0, G0=0
Sort(c4)･･･S_sum = 0.7677, lnA = -154.0, T2 = 70.0, G0=0
最適値；sort(c2)→s047.f90と同じ。
・バッチ形式
Sort(c1)･･･S_sum = 0.2274, lnA = -50, T2 = 12.0, G0=0
Sort(c2)･･･S_sum = 0.1273, lnA = -56.0, T2 = 20.0, G0=0
Sort(c3)･･･S_sum = 0.5476, lnA = -102.0, T2 = 54.0, G0=0
Sort(c4)･･･S_sum = 0.7677, lnA = -154.0, T2 = 70.0, G0=0
```

Fig. 19　Sections18.f90 におけるサーチ結果. バッチ形式で値が一致しなかった原因は不明である.

並列化したプログラムの計算時間が短縮されたかを確認するため, 再びサーチを行った. 同じ条件, 同じ範囲でサーチした結果を比較する. 全てスーパーコンピュータのシステム B を利用した. 用いた条件は, T_0=160, T_a=94, T_b=20, T_c=160, T_g=102, Q_1=-12, Q_2=-9.5, Q_3=5, β=0.330, B=1000 であり, 設定した範囲は, -200<lnA<0(キザミ 2), -20<T2<70(キザミ 2), 0<G0<0.15(キザミ 0.03)とした. 会話型における計算時間には限界の時間があったため, 短めの時間となるように周回数を設定した. 結果を Table 1 に示す.

Table 1　4並列における時間短縮の結果.

	時間	ln*A*	T_2	G_0	sum	偏差
会話型	12min46sec	-54	18	0	0.12134	0.142
バッチ形式	2h23min57sec	-54	18	0	0.12134	0.142
並列化なし	2h55min36sec	-54	18	0	0.12134	0.142

並列化は 4 並列であり, 並列化なしはバッチ形式で行った. 表に示した時間は, a.out の作成時刻と output の作成時刻の差から判断している. バッチ形式の計算時間が, 会話型に比べて長いのは, バッチ処理の待機時間が長い事が原因である.

3.　議論と応用

3.1 10Sections.f90(10 並列)

4 並列では, 大きな時間短縮が見られなかったため, 10 並列を目標にした. 10Sections.f90 では, サブルーチンとして呼び出す sort の数を 4 つから 10 へ増やした. バッチ形式を行うための, スクリプト内のスレッド数に注意しt=16とした. 問題なく10並列を組むことに成功した. 並列化なしと 4 並列, 10 並列における時間短縮の結果を Table 2 に示す.

Table 2　10並列における時間短縮の結果.

プログラムの種類	所要時間
並列化なし	29h28min32sec
4 並列	22h58min22sec
10 並列	10h18min48sec

1/10倍とまではいかなかったが, 目に見えて速くなった事が分かる.

3.2 自動並列 autoParallelising(並列化プログラム)

S_sum の自動算出, 最適値の自動出力を目標とする. 現在用いている 10Sections.f90 は, それぞれサブルーチンごとに最適値を出力し, 最終的には目視で最適値を選んでいる. ここでは, 目視で選んできた部分をプログラム上で行えるようにシステムを改良した. 作戦としては, ①メインにて sum(:), lnA(:), T2(:), G0(:)のような割り付け変数を定義する. ②サブルーチン内でそれぞれ 1～10 の数字で最適値を保存する. (Sort1 では sum(1)=S_sum, lnA(1)=lnA, T2(1)=T2, G0(1)=G0 とする. Sort2 では…以下略)③メインに戻った後に割り付けできているか確認する. ④割り付けできていれば, if 文を用いて最適値の決定を行う. 以上の手順が妥当であると考えた. 行う上での課題となる点は, 同じソースコードを用いているサブルーチンで, どのようにして各 sort に異なる割り付けをするかである. sort1 で ss=1, sort2 で ss=2 のようなシステムを組む方法を考えた. この解決策としても if 文を用いる事を使用した. 各 sort で異なる箇所は x の値だけである. この x を利用して ss という変数に対して異なる値を与えていった. 詳細を Fig. 20 に示す.

```
if (x == c1) then
ss=1 ; end if
if (x == c2) then
ss=2 ; end if
if (x == c3) then
ss=3 ; end if
```

Fig. 20　自動並列を行う上での, 割り付け変数の決定. 各 sort(x)が x=c1, c2, c3 という異なる値を持つ事を利用した.

この方法により，各 sort における異なる変数を割り当てることに成功した．あとは，これらを S_sum(ss), Sum_A(ss), Sum_T2(ss), Sum_G0(ss)として保存し，サブルーチンから本文に戻ったところで，さらなる最適値決定の if 文による比較を行った．自動並列を行う上での概要を付録の 3 と 4 として示す．

実際に作成した 3 並列における自動並列のプログラムを 3parallevol4.f90 とした．3parallevol4.f90 では，最適値を output.d(出力ファイル)に書き出し，出力することが可能となった．

4. 結　言

以上の内容にて，本研究の並列化プログラミングの目的が全て達成されたことになる．当初の目的とされていた時間短縮は，およそ3倍の短縮に成功した．この成果によって，今まで実践しづらかった長時間の研究も可能となった．

5. 参考文献

[1] Y.Okuya and Y.Tanaka : Mem. Grad. Eng. Univ. Fukui, 62:63, 9(2014).

[2] Tool AQ : *J Am Ceram Soc*, 29, 240(1946) .

[3] Narayanaswamy OS : *J Am Ceram Soc*, 54, 491 (1971).

[4] Moynihan CT, Macedo PB, Montrose CJ, Gupta PK, DeBolt MA, Dill JF, et al : *Anm NY Acad Sci*, 279, 15(1976) .

[5] J. L.Gomez Ribelles and M. Monleon Pradas : *Polymer*, 38-4, 963(1997).

[6] Adam, G. and Gibbs, J. H : *J. Chem. Phys*, 43, 139 (1965) .

[7] Gibbs, J. H. and DiMarzio, E. A : *J. Chem. Phys,* 28, 373(1958).

[8] http://www.nag-j.co.jp/openMP/