スーパーコンピュータ SR11000 でのプログラム開発事例：カップルドクラスター法による高分子の電子状態計算

片 桐 秀 樹

産業技術総合研究所　計算科学研究部門

１．はじめに

本稿では，**カップルドクラスター(coupled-cluster)法**という**電子状態計算**の手法を，高分子に適用するために行なったスーパーコンピュータ SR11000 でのプログラミングについて，その一部を紹介したいと思います。その前に，電子状態計算とは何なのか，そこから話を始めることにしましょう[1-3]。

我々の世界に存在する物質は，原子の集合体でできています。さらに，原子は核と電子から成り立っています。電子と核の間にはクーロン力による引力が働き，また，電子と電子の間は斥力が働いています。通常，電子は核に束縛された安定な状態にありますが，光と相互作用したり，原子や電子が衝突することによって，その状態は変化します。電子が作り出す様々な状態を**電子状態**と呼んでおり，我々は物質の電子状態を知ることによって，その物質の電気的・光学的性質や化学反応などの化学的性質を理解または予測することができます。

さて，電子状態を理論的に調べることとは，電子の**波動関数**を求めること，すなわち電子の**シュレディンガー方程式**を解くことに他なりません。量子力学が誕生して以来，電子状態計算の方法論は大きく発展しました。特に，コンピュータの進歩によって電子状態計算は格段の進歩を遂げ，物質科学に多大な貢献をするに至っています。しかし，例えば水分子のような電子を 10 個しか含まないような単純な系であっても，その波動関数を正確に求めることは，現在でも困難な問題です。これは，電子間に働くクーロン相互作用が長距離力であり，この相互作用のもとに多体問題を解くことが容易ではないことが関係しています。そのため，近似的に波動関数を求める様々な手法が提案されています。

現在，電子状態計算に広く使われている方法はいくつかあります。（但しここでは，経験的なパラメータを用いたモデル理論による方法は除き，そのようなパラメータを必要としない，いわゆる**第一原理計算**とか **ab initio 計算**と呼ばれるものに限定することにします。）

一つは，**ハートリーフォック(Hartree-Fock)法**と呼ぶ方法によって波動関数を求める方法，そしてそれに基づいてさらに精度の高い波動関数を求める方法です。ハートリーフォック法では，**平均場近似**（つじつまの合った場の方法）を用いてシュレディンガー方程式を解き，独立粒子の波動関数を求めます。これで用が足りる場合もありますが，より精度を求める場合や，基底状態だけではなく励起状態を求めたい場合は，さらに進んで**多体効果**を取り入れる必要があります。この多体効果のことを**電子相関**と呼んでいます。電子相関を取り入れる方法としては，配置間相互作用法，摂動法，また，本稿で取り上げるカップルドクラスター法，などがあります。これらの方法は，従来から主に原子・分子に適用されながら発展してきましたが，計算コストが高いことがネックとなり，固体への応用例は多くはありません。

もうひとつの有力な方法は，**密度汎関数法**と呼ばれる方法です。これは，**ホーヘンバーグ・コ**

ーン（Hohenberg-Kohn）の定理に基づき，系のエネルギーが電子密度の汎関数となっていることを利用して，電子状態を求める手法です。汎関数の形は未知なので厳密な解を求めることはできませんが，便宜的に**コーン・シャム（Kohn-Sham）方程式**を解く方法が使われます。密度汎関数理論では，電子の多体効果は，電子密度の関数で表される**交換相関ポテンシャル**によって取込むことができます。密度汎関数法は，アボガドロ数のオーダーとなる数の電子を含む凝縮系の計算に適しており，固体のバンド計算の分野で大きな成功をおさめました。分子計算の分野においても近年急速に普及しています。

密度汎関数法は，ハートリーフォック法に基づく方法に比べ，少ない計算量の割に精度の高い結果が得られるという利点があるのですが，少なからず問題点があることもわかってきています。それは，バンドギャップを過小評価することや，強相関電子系などいくつかの系に対して正しい計算結果が得られないという点です。これらの問題点を改善すべく，交換相関ポテンシャルを改良する方法や，GW 近似などの方法などが，現在精力的に研究されています。

これら以外の電子状態を研究する有力な方法としては，**量子モンテカルロ法**があります。この方法は，系のサイズを増やしたときに計算量が極端に増加しないこと，並列コンピュータに実装しやすいことなどのメリットがあり，今後有望な方法の一つです。

さて，筆者は，ハートリーフォック法およびカップルドクラスター法を固体（一次元高分子）に適用する研究[4]を行なっており，そのためのプログラム開発を行なっています。カップルドクラスター法は，ハートリーフォック法から出発した後に電子相関を考慮する方法の一つで，精度が高く，分子計算の分野ではすでに標準的な手法として使われています。しかし，カップルドクラスター法は非常に大きな計算量を必要とするため，系のサイズが大きい時に解くことは現在でも容易ではありません。このことは特に，固体に応用する場合には大きな問題になります。カップルドクラスター法を固体などの大きな系に適用するためには，高速なコンピュータの利用は必須であると言えるでしょう。

次節以降ではカップルドクラスター法の一般論を述べてから，高分子に適用する場合の具体的な計算手順を，計算量の大きい部分に重点を置いて説明します（２，３節）。ここでは読者に，ある程度の量子力学の知識があることを想定しています。後半では，カップルドクラスター法の計算で大きなウエイトを占める大次元配列の積和計算に焦点を当て，スーパーコンピュータでのプログラミングのテクニックを解説します（４節）。

２．カップルドクラスター法とは何か

カップルドクラスター法は上で説明したように，**一粒子の波動関数（ハートリーフォック軌道）**を出発点として，多体効果を取込んだ波動関数を求める方法です。この方法は Coester らによって核物理学における多体問題に使われたのが始まりで[5]，その後，Cizek によって電子の計算に応用されました[6]。

カップルドクラスター法では，まず最初にハートリーフォック法で次の**固有値方程式**を解き，ハートリーフォック軌道を求めます。

$$\hat{F} = \hat{h} + \sum_{j=1}^{N} (\hat{J}_j - \hat{K}_j) \qquad (1)$$

$$\hat{F}|\phi_i\rangle = \varepsilon_i |\phi_i\rangle \qquad (2)$$

$\hat{F}$ は**フォック演算子**で，$\hat{h},\hat{J},\hat{K}$ はそれぞれ**一電子**（電子の運動エネルギー項と電子-核の引力項の和），**クーロン**，**交換**の各相互作用を表す演算子を表します。Nはここでは電子数を表します。ϕ_iが求めるべきハートリーフォック軌道で，ε_iは対応するエネルギー固有値です。ハートリーフォック法では，個々の電子が他の電子が作る平均的な場の中を運動しているという近似のもとに電子の運動を記述します。ϕ_iを適当な**基底関数**の展開で表せば，(2)式は行列の固有値問題になり，容易に解くことができます。基底関数としては，$\hat{h},\hat{J},\hat{K}$ の行列要素が簡単に計算できるという理由で，ガウス型関数 $\exp(-\alpha r^2)$ の線形結合で表した関数が使われます。これは**ガウス型基底関数**と呼ばれ，分子の電子状態計算で広く用いられています。ガウス型基底関数は周期律表の各原子に対して決められたものが多数あり，容易に入手することができます[7]。

さて，カップルドクラスター法は，**多体波動関数**が以下の様な形で書けるという ansatz[†]に基づいています。

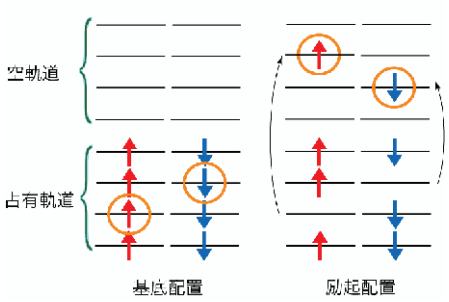
$$\left|\Psi_{CC}\right\rangle = \exp(\hat{T})\left|\Psi_0\right\rangle \qquad (3)$$

$\left|\Psi_0\right\rangle$は上で求めたハートリーフォック軌道関数$\{\phi_1,\phi_2,\phi_3,\cdots\}$から作られた，反対称化された**スレーター行列式**$\|\phi_1\phi_2\phi_3\cdots\|$です。$\hat{T}$ は**励起演算子**と呼ばれ，さらにプリミティブな励起演算子の和 $\hat{T}=\hat{T}_1+\hat{T}_2+\hat{T}_3+\cdots$で表されます。

励起演算子は，ハートリーフォック法で求めた**占有軌道**（電子が入った軌道）から**空軌道**（電子が空の軌道）に電子を励起させる働きを持ちます。**生成・消滅演算子**を用いて$\hat{T}_1,\hat{T}_2,\hat{T}_3,\cdots$を具体的に表せば，以下のようになります。

$$\hat{T}_1=\sum_{ia} t_i^a a^\dagger i,\quad \hat{T}_2=\frac{1}{(2!)^2}\sum_{ijab} t_{ij}^{ab} a^\dagger i b^\dagger j,$$
$$\hat{T}_3=\frac{1}{(3!)^2}\sum_{ijkabc} t_{ijk}^{abc} a^\dagger i b^\dagger j c^\dagger k,\quad \cdots \qquad (4)$$

ここで，$i,j,k,\cdots$は被占軌道に対する消滅演算子，$a^\dagger,b^\dagger,c^\dagger,\cdots$は空軌道に対する生成演算子です。$t_i^a,t_{ij}^{ab},t_{ijk}^{abc},\cdots$は後に述べる**カップルドクラスター方程式**を解いて求められる係数で，それぞれ一電子励起，二電子励起，三電子励起に対応する係数です。二電子励起の例を第 1 図に示します。



第 1 図：二電子励起配置の例。

矢印はハートリーフォック軌道を占有する電子を表し，赤と青はスピンの向きに対応しています。

[†]仮説（仮設と訳す場合もあります[8]）。

カップルドクラスター法では，このようなハートリーフォック軌道間の電子励起を考慮することにより，平均場近似を越えた多体効果を取り入れることができます。

カップルドクラスター法の利点は，(3)の形の多体波動関数が，**size-extensivity** を満足することにあります。Size-extensivity とは，系のサイズを大きくしていったときに，近似の程度が変わらない性質のことです。例えば，相互作用のない N 個の系を一つとみなして計算して得られるエネルギーが，個別に計算して得られるエネルギーの和(つまり 1 個のエネルギーの N 倍)となっていれば，size-extensivity を満たしていると言います。この性質は固体の計算などの電子数が多い系の計算では重要です。

さて，$t_i^a, t_{ij}^{ab}, t_{ijk}^{abc}, \cdots$は，以下の時間に依存しないシュレディンガー方程式を解くことによって決定されます。

$$\hat{H}\exp(\hat{T})\left|\Psi_0\right\rangle = E\exp(\hat{T})\left|\Psi_0\right\rangle \qquad (5)$$

ここで $\hat{H}$ は第二量子化で表された電子系のハミルトニアン，E はエネルギー固有値です。この方程式の左から $\exp(-\hat{T})$ を掛け，さらに一電子励起関数 $\left|\Psi_i^a\right\rangle$，二電子励起関数 $\left|\Psi_{ij}^{ab}\right\rangle$，三電子励起関数 $\left|\Psi_{ijk}^{abc}\right\rangle$, …を射影すると次の連立方程式(カップルドクラスター方程式)が得られます[9]。

$$\left\langle\Psi_i^a\left|\exp(-\hat{T})\hat{H}\exp(\hat{T})\right|\Psi_0\right\rangle = 0, \qquad (6)$$

$$\left\langle\Psi_{ij}^{ab}\left|\exp(-\hat{T})\hat{H}\exp(\hat{T})\right|\Psi_0\right\rangle = 0, \qquad (7)$$

$$\left\langle\Psi_{ijk}^{abc}\left|\exp(-\hat{T})\hat{H}\exp(\hat{T})\right|\Psi_0\right\rangle = 0, \qquad (8)$$

…

これらの方程式の左辺を展開すると，係数 $t_i^a, t_{ij}^{ab}, t_{ijk}^{abc}, \cdots$（t と略記）とハートリーフォック軌道を基底にとって表した**電子間反発積分（二電子積分）**の積で表される沢山の項の和が出てきて，係数 t に対する連立非線形方程式になっていることを示すことができます。このことについては後でもう一度触れます。この方程式を解く方法としては，得られた式をさらに t に対する漸化式に変形し，逐次法を用いて t を求める方法が用いられます。このようにカップルドクラスター法は，**変分原理**に基づいてシュレディンガー方程式を解く他の多くの電子状態計算手法とは大きく異なっています。

係数 t，すなわち $\hat{T}$ が決まれば，系の基底状態のエネルギーを次式で求めることができます。

$$\left\langle\Psi_0\left|\exp(-\hat{T})\hat{H}\exp(\hat{T})\right|\Psi_0\right\rangle = E \qquad (9)$$

さて，励起演算子 $\hat{T}$ は，一電子励起，二電子励起，三電子励起，…というように，系の電子の数に応じて無限項の和になります。もし，可能な全ての励起演算子を考慮できれば，厳密解に到達することができると考えられる訳ですが，電子が数個程度の場合ならばともかく，一般の場合はそうはいきません。幸い二電子励起まで考慮すれば，かなり精度の高い波動関数が得られることがわかっているので，$\hat{T}_3$ 以降を打ち切る近似（**CCSD 近似**と呼ぶ）あるいはその派生形がよく

用いられます。その場合でも，求めなければいけない係数 t の数は相当の数になります。例えば，被占軌道及び空軌道の数をそれぞれ 100 個とした場合，t_{ij}^{ab} の数は 10e+8 個となり，t_{ij}^{ab} に対する連立方程式の数も同じだけ必要になります。そんなにたくさんの未知数があって，しかも非線形になっているこの方程式の解が無事に求まるのかどうか心配になりますが，多くの場合，逐次法で解くときちんと収束します。ただしそのためには，ハートリーフォックの波動関数から作られたスレーター行列式が，多体波動関数の十分によい近似になっている必要があります。

さて，カップルドクラスター法では励起状態を求めることもできます。いくつかの方法が提案されていますが，ここではそのうちの一つである equation-of-motion (EOM) という方法[9]に従って説明します。まず，(6)式の左辺のブラケットの中にある演算子 $\exp(-\hat{T})\hat{H}\exp(\hat{T})$ を一電子励起関数，二電子励起関数，三電子励起関数，…等々を基底にとることによって行列表示してやると，係数 t の要素の数と同じ次数をもつ巨大な非対称行列が得られます。励起状態はこの行列の固有値問題を解くことによって求めることができます。この行列の各々の要素は，カップルドクラスター方程式と同様に，係数 t と二電子積分の間の積和で表されるたくさんの項の和で表されるので，非常に多くの計算量が必要になります。

ここまではカップルドクラスター法の一般論を述べてきました。次節では，カップルドクラスター法を一次元系（高分子）に適用する場合について，具体的な計算手順を述べます。

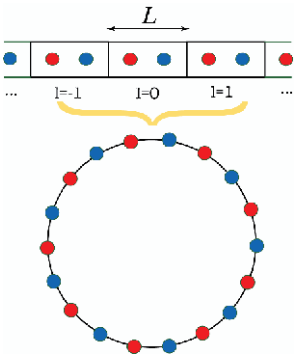
３．カップルドクラスター法の一次元系への適用

カップルドクラスター法を一次元系に適用するには，もちろんまず最初に，一次元系のハートリーフォック軌道（周期系なので**結晶軌道**とも呼ぶ）を求める必要があります。周期的境界条件を満足するハートリーフォック軌道の波動関数は，**波数**を k，軌道の番号を i とすると次の形で書けます[10]。

$$\varphi_i^k(\mathbf{r}) = \frac{1}{\sqrt{N_C}} \sum_{l=-l\max}^{l\max} e^{ikl} \sum_p C_{pi}^k \chi_p^l \qquad (10)$$

ここで χ_p^l は l 番目のセルに置かれた p 番目のガウス型基底関数を表し，また，C_{pi}^k は結晶軌道の展開係数（複素数）を表します。上式が普通の原子や分子の計算と異なっている点は，波数 k が新たに加わり，それに依存する e^{ikl} という位相因子がかかること，それから，格子の番号 l に対する和が新たに加わることです。

式(10)の N_C はセルの数を表しており，これは第２図に示すように，波動関数を周期的に並んだ有限個のセルで表現していることによるもので，式(10)の場合では $N_C = 2l_{\max} + 1$ となります。またこのことに対応して，波数 k は離散的な値，すなわち $k = 2\pi j / L / N_C$ として，$j = 0,1,2,\mathrm{L}\ ,N_C - 1$ の値しかとることができません（L は単位セルの長さ）。



第２図:一次元周期的境界条件の模式図。

ガウス型基底関数を用いて一次元周期系の計算を行なう場合には，ある有限の個数（N_C）の単位セル（長さ L）が仮想的に円周上に並んでいると見立てて周期的境界条件を表現します。上の図は $N_C = 9$ （$l_{\max} = 4$）の場合です。赤と青の丸は原子を表しています。

結晶軌道を求めるには，フォック演算子をガウス型基底関数で行列表示したもの（**フォック行列**）と基底関数の**重なり積分行列**を，波数 k 毎に求める必要があります。それらを以下に示します。

$$F_{rs}^{k} = \sum_{l} e^{ikl}\left(h_{rs}^{0l} + J_{rs}^{0l} - K_{rs}^{0l}\right) \qquad (11)$$

$$S_{rs}^{k} = \sum_{l} e^{ikl} S_{rs}^{0l} \qquad (12)$$

$h_{rs}^{0l}, J_{rs}^{0l}, K_{rs}^{0l}$ は，フォック演算子の一電子項，クーロン項，交換項を，それぞれガウス型基底関数 χ_r^0 と χ_s^l の行列要素として表したもので，S_{rs}^{0l} は重なり積分 $\int \chi_r^0(\mathbf{r})\chi_s^l(\mathbf{r})d\mathbf{r}$ です。

これらの中で計算量が最も多いのは，J_{rs}^{0l} および K_{rs}^{0l} の部分です。これらは密度行列 $P_{tu}^{l} = \frac{2}{N_C}\sum_{i}^{occ}\sum_{k}^{BZ} C_{it}^{k*} C_{iu}^{k} e^{ikl}$ と二電子積分の積になっており，次式のように計算されます。

$$J_{rs}^{0l} = \sum_{mn}\sum_{tu} P_{tu}^{n}(r^0 s^l \mid t^m u^n) \qquad (13)$$

$$K_{rs}^{0l} = \frac{1}{2}\sum_{mn}\sum_{tu} P_{tu}^{n}(r^0 t^m \mid s^l u^n) \qquad (14)$$

ここで二電子積分は以下のように定義されます。

$$(r^0 s^l \mid t^m u^n) \equiv \int \chi_r^0(\mathbf{r}_1)\chi_s^l(\mathbf{r}_1)\frac{1}{r_{12}}\chi_t^m(\mathbf{r}_2)\chi_u^n(\mathbf{r}_2)d\mathbf{r}_1 d\mathbf{r}_2 \qquad (15)$$

式(13)，(14)は r,s,l の各組に対して m,n,t,u の四重ループを回す**積和演算**になっています。全部で７重のループになるので一見大変そうですが，並列プログラミングを用いて容易に並列化することができます。例えば，（15）式の二電子積分を (l,m,n) のラベルで分割して複数のプロセッサに割当て，式(13)，(14)の計算をそれぞれのプロセッサで独立に行ない，最後に，各プロセッサの J_{rs}^{0l} および K_{rs}^{0l} を足し合わせれば計算が完了します。

フォック行列が求まれば，あとはフォック行列と重なり積分行列の一般化固有値問題を解くことによって，ハートリーフォック結晶軌道を求めることができます。この固有値問題の規模は，計算の対象となる系のサイズをNとする時，N^3に比例して大きくなりますが，カップルドクラスター方程式を解く部分に比べればずっと小さい計算量なので，大した問題ではありません。

カップルドクラスター法の計算に進むためには，二電子積分をハートリーフォック結晶軌道で表したものに変換する必要があります。このプロセスのことを**積分変換**と呼びます。以下にその二電子積分の定義と変換式を示します[4, 11-13]。

$$\left\langle i(k_1)j(k_2)\middle|k(k_3)l(k_4)\right\rangle \equiv \int \phi_i^{k_1*}(\mathbf{r}_1)\phi_j^{k_2*}(\mathbf{r}_2)\frac{1}{r_{12}}\phi_k^{k_3}(\mathbf{r}_1)\phi_l^{k_4}(\mathbf{r}_2)d\mathbf{r}_1 d\mathbf{r}_2$$

$$= \frac{1}{N_C}\delta_{T(k_1+k_2),T(k_3+k_4)}\sum_{l_2 l_3 l_4} e^{-ik_2 l_2}e^{ik_3 l_3}e^{ik_4 l_4} \times \sum_{rstu} C_{ir}^{k_1*}C_{jt}^{k_2*}C_{ks}^{k_3}C_{lu}^{k_4}(r^0 s^{l_3}|t^{l_2}u^{l_4}) \quad (16)$$

上の式の中に現れる$T(k_1+k_2)$は，k_1+k_2を**第一ブリルアンゾーン**に還元するための関数です。上の式は，波数k_1+k_2とk_3+k_4とがそれぞれ第一ブリルアンゾーンに還元したときに，等しくなっていない場合には0になります。

ここで，二電子積分の略記法について触れます。式(15)と(16)は，どちらも二電子積分を表す略記法ですが，よく見ると$\mathbf{r_1}$と$\mathbf{r_2}$の順番が違っていることがわかると思います。一般に，(15)のような二電子積分の記法を Mulliken's notation，(16)のような記法を Dirac's notation と呼び，括弧の形を変えることで区別しています。なお，(16)式では，ブラとケットの二つのϕ_i^kの積は反対称化されていませんが，反対称化された積（スレーター行列式）を使いたい場合があります。その場合は式(16)の略記法に代えて，$<i(k_1)j(k_2)\|k(k_3)l(k_4)>$のように表します。これはカップルドクラスター方程式等を記述するときに使います。

さて，式(16)の計算もフォック行列の計算と同様，かなりループのネストが深い計算となっており，i,j,k,l,k_1,k_2,k_3,k_4の組み合わせに対して，r,s,t,u,l_2,l_3,l_4の7重ループの積和計算が必要です。しかし，こちらもフォック行列の計算と同様，簡単に演算を並列化することができます。式(16)は全体にδ関数がかかっているため，全ての波数kの組み合わせの項を計算する必要はなく，$(N_C)^3$個の項しか値を持ちません。そこで，値を持つ波数kの組み合わせをテーブル化しておき，それをいくつものプロセッサに分割して割り当てて式(16)を分担して計算させれば，簡単に並列計算を実現できます。なお，フォック行列と積分変換の計算は，どちらも並列化効率は非常に良好です。

ハートリーフォック結晶軌道と，それによって表した二電子積分が求まれば，カップルドクラスター法の計算に進むことができます。ところで，筆者が最初に作ったカップルドクラスター法のコードは，分子の計算のために書いたものでした。分子の場合，軌道の展開係数と二電子積分は実数となり，複素数である式(16)をそのまま適用することはできません。幸い，波数kと$-k$の結晶軌道の展開係数は互いに複素共役の関係にありますので，それら結晶軌道のユニタリー変換（和と差）を作って結晶軌道を再定義し，これらを基底にとって式(16)の二電子積分を再度変換すれば，二電子積分は実数となります[4]。この方法は，波数kがもつ対称性を計算に利用することができなくなる欠点がありますが，分子用に作られたコードをそのまま用いることができるという便利さがあるために，筆者はこの方法を用いています。

以下の説明では，式(16)の二電子積分に出てくる$i(k)$の波数kの表示を省略し，一つのシンボル（i,j,a,bなど）で軌道番号と波数のペアを表すことにします。

さて，それではカップルドクラスター方程式の計算に進みます。この方程式はCCSD近似の場合，式(6),(7)だけで表されます。ハミルトニアン$\hat{H}$を（反対称化された）二電子積分を用いて**第二量子化表示**で表した式：

$$\hat{H} = \sum_{pq} h_{pq} p^{\dagger} q + \frac{1}{4} \sum_{pqrs} \langle pq || rs \rangle p^{\dagger} q^{\dagger} sr \qquad (17)$$

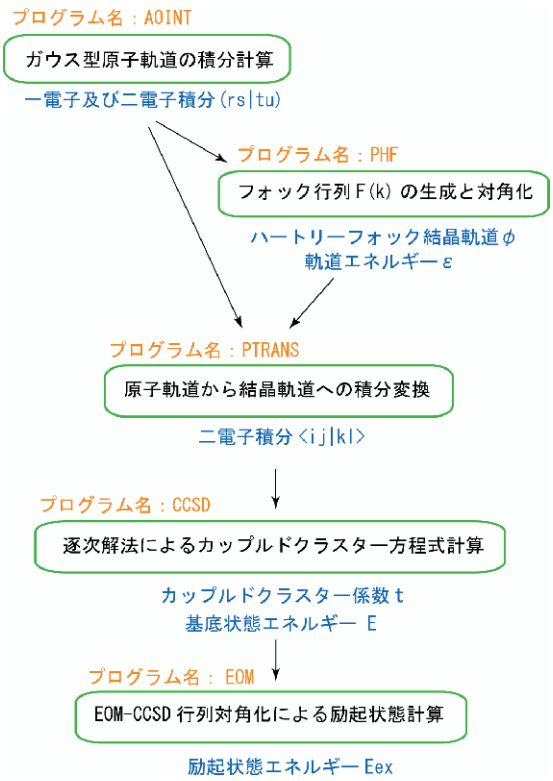
を式(6),(7)に代入して展開すると，二電子積分$< pq \parallel rs >$，係数t_i^a，t_{ij}^{ab}の積でできた多数の項の和で表された非線形連立方程式が得られます。この方程式を逐次法による繰り返し計算によって解くのが，カップルドクラスター計算の主要部分になります。これらの方程式は非常に長い式になるので，そのうちの一つの項だけを挙げるとこんな風です。

$$\sum_{mef} \langle am || ef \rangle t_i^f t_{mj}^{eb} \qquad (18)$$

これは，(6) 式から出てくる項のほんの一部です。この式も実際には，i,j,a,bとm,e,fが軌道の番号のほかにスピンの自由度を持っていますので，スピンをあらわに表示すると，さらに複雑な式になります。カップルドクラスター方程式に現れるほとんどの項の基本的な構造は，上と同様の形，すなわち，二電子積分と係数ｔの積という形になっていて，異なっているのは，積和を取る添字の位置や積和をとる係数ｔの種類や数です。これらは多体論で使われるダイアグラムを用いて表すこともできます。

さて，式(18)の計算は，m,e,fに関する積和計算を全てのi,j,a,bの組み合せに対して行なう必要があります。計算量は，結晶軌道と波数の数の積をNとすると，だいたいN^6に比例します。また，二電子積分やt_{ij}^{ab}には，N^4のオーダーの記憶量が必要になります。式(18)の計算は，i,j,a,bの組それぞれにおいて独立となっているので，一見，並列化が容易なように見えるのですが，二電子積分やt_{ij}^{ab}へのアクセスのパターンが複雑で，しかもそれが項によって大きく異なること，これらのパーツ及び積和の結果を格納しておく一時的な領域が，非常に大きなメモリを消費することの二つの相乗効果で，実際にはそれほど単純ではありません。次節では式(18)の実際のコーディング例を示し，それらの性能を比較することにします。

この節を終わる前に，第3図に筆者が作成したカップルドクラスター法のプログラムの全体図を示します。プログラムは全てFORTRANで記述し，並列化はOpenMPを用いて行っています。OpenMPは**共有メモリ型の並列コンピュータ**上で利用することができますが，一般には，PCを集めたいわゆる**クラスター型の並列コンピュータ**では利用することが出来ないので注意が必要です。OpenMPは，プログラムに数行の指示行を加えるだけで並列化が出来るため，作業効率が良く，さらに，指示行の挿入でプログラムの意味が変わることがないという利点があり，筆者は気に入っています。OpenMPの使用法の詳細については，本特集号の記事をご覧ください。また，SGI社のチュートリアル等の資料[14]も分かり易くてお勧めです。



第３図：　カップルドクラスター法プログラムの全体図。

４．カップルドクラスター法における積和計算のプログラミング

この節では，カップルドクラスター法で必要となる積和計算について，式(18)の計算（正確にはその一部）に焦点を当てて，プログラミング例を紹介することにします。（なお，以下に挙げるプログラムは本稿のために，元のプログラムに手を加えて修正したものです。）

式(18)をスピンの種類をあらわに表示した書き方に改め，その一部を取り出したものを以下に示します。

$$w_{i\bar{j}}^{a\bar{b}} = w_{i\bar{j}}^{a\bar{b}} + 2\sum_{mef}(af|em)t_i^f\left(t_{m\bar{j}}^{e\bar{b}} + t_{mj}^{eb}\right) - 2\sum_{mef}(ae|fm)t_i^f t_{m\bar{j}}^{e\bar{b}} \qquad (19)$$

$$w_{i\bar{j}}^{b\bar{a}} = w_{i\bar{j}}^{b\bar{a}} - 2\sum_{mef}(ae|fm)t_i^f t_{m\bar{j}}^{b\bar{e}} \qquad (20)$$

これらの式はカップルドクラスター係数 t を求める収束計算のサイクルで，毎回一度実行されます。各イタレーションで，t に初期値を入れて上の式を計算し，さらに他の項も同様の計算をして，最終的に得られた $w_{i\bar{j}}^{a\bar{b}}$ が新しい $t_{i\bar{j}}^{a\bar{b}}$ の初期値となります。上の式でバーの無し，有りはスピンの上向きと下向きに対応しています。t_{ij}^{ab} と $t_{i\bar{j}}^{a\bar{b}}$ はスピンをあらわに表示したために出てきたもので，両者は全く異なるものであることに注意してください。$(pq|rs)$ の形をしているものは，Mulliken の記法による二電子積分です。また，i,j,m,n は占有軌道，a,b,e,f は空軌道に対する添字を表しています。以下ではそれぞれの軌道の数を NOCC,NVIR とします。

式(19),(20)は，二電子積分$(pq|rs)$とt_i^aとt_{ij}^{ab}（または$t_{\bar{i}\bar{j}}^{a\bar{b}}$）の積和をとり，$w_{\bar{i}\bar{j}}^{a\bar{b}}$に加算または減算する形になっています。ここで注意しなければならないことは，この計算は２段階に分けて実行する必要があるという点です。例えば式(20)を計算する場合，以下のようにしてはいけません。

```
do i=1,NOCC; do j=1,NOCC; do a=1,NVIR; do b=1,NVIR;
```

$$w_{\bar{i}\bar{j}}^{b\bar{a}} = w_{\bar{i}\bar{j}}^{b\bar{a}} - 2\sum_{mef}(ae|fm)\times t_i^f \times t_{m\bar{j}}^{b\bar{e}}$$

```
enddo; enddo; enddo; enddo;
```

プログラム１：ダメなループの回し方

この方法では全体として７重のループとなり，計算量はNOCC^3*NVIR^4となってしまいます。これは以下のように計算する方が得です。

```
do a=1,NVIR; do e=1,NVIR; do i=1,NOCC; do m=1,NOCC;
```

$$A(a,e,i,m) = 2\sum_{f}(ae|fm)\times t_i^f$$

```
enddo; enddo; enddo; enddo;
do i=1,NOCC; do j=1,NOCC, do a=1,NVIR;, do b=1,NVIR;
```

$$w_{\bar{i}\bar{j}}^{b\bar{a}} = w_{\bar{i}\bar{j}}^{b\bar{a}} - \sum_{me}A(a,e,i,m)\times t_{m\bar{j}}^{b\bar{e}}$$

```
enddo; enddo; enddo; enddo;
```

プログラム２：　正しいループの回し方

この方法だと前半のループは５重，後半が６重となり，後半部分の計算量は NOCC^3*NVIR^3 となります。NOCC や NVIR は計算する系の大きさによっては数百に達しますので，計算量は最初のやり方に比べて圧倒的に少なくてすみます。掛け算の順序は，t_i^aを先に掛ける方法のほかに後で掛ける方法もあり，その場合の計算量は NOCC^2*NVIR^4 となります。どちらが得かは NOCC と NVIR の大小関係に依存します。一般には NOCC<NVIR となっていることが多いため，上で示したようにt_i^aを先に掛ける方が計算量が少なくなります。

さて，上の計算は前半・後半ともに単なる行列の積で表すことができますので，線形演算の標準パッケージである BLAS ライブラリを使って計算はおしまい，と言いたいところですが，ワーク配列である A は非常に大きな記憶領域(例えば NOCC=NVIR=256 の時で 32GB)を占めるので，少しでもメモリを節約するためには何らかの工夫が必要です。そこで以下では５つのプログラム例を示し，その性能を比較することにします。

４．１　ワーク配列を全く使わないプログラム(SIMPLE.f)

まず，ワーク配列を全く使わない例をお見せします。

プログラム例1： SIMPLE.f

```
 1      do mm=1,NOCC
 2        do me=1,NVIR
 3 !$OMP PARALLEL DO DEFAULT(PRIVATE),SHARED(ia,idx,t1,t2s,t2a,w2a,v,
 4 !$OMP& me,mm)
 5          do mi=1,NOCC
 6            do ma=1,NVIR
 7              mea=idx(ma+NOCC,me+NOCC)
 8              sum1=0d0
 9              sum2=0d0
10              do mf=1,NVIR
11                mem=ia(me+NOCC)+mm
12                mfm=ia(mf+NOCC)+mm
13                mfa=idx(mf+NOCC,ma+NOCC)
14                sum1=sum1+v(mfa,mem)*t1(mf,mi)
15                sum2=sum2+v(mfk,mem)*t1(mf,mi)
16              enddo
17              do mj=1,NOCC
18                do mb=1,NVIR
19                  w2a(mb,ma,mj,mi)=w2a(mb,ma,mj,mi)
20     &                          +2.0d0*(sum1*(t2s(mb,me,mj,mm)
21     &                                      +t2a(mb,me,mj,mm))
22     &                                -sum2*t2a(mb,me,mj,mm))
23                  w2a(ma,mb,mj,mi)=w2a(ma,mb,mj,mi)
24     &                          -2.0d0*sum2*t2a(me,mb,mj,mm)
25                enddo
26              enddo
27            enddo
28          enddo
29        enddo
30      enddo
```

このプログラムでは二電子積分$(pq|rs)$を2次元配列v(pq,rs)に，$t_i^a, t_{ij}^{ab}, t_{\bar{i}\bar{j}}^{a\bar{b}}, w_{\bar{i}\bar{j}}^{a\bar{b}}$を2次元及び4次元配列 t1(mi,ma),t2s(mb,ma,mj,mi),t2a(mb,ma,mj,mi),w2a(mb,ma,mj,mi)に，それぞれ割り当てています。このプログラムではvとt1の積をsum1,sum2にキープすることで，ワーク配列を使わずに済ませています。3,4行目はOpenMPの指示制御文で，ここではループ変数miについて並列化しています。外側のmeまたはmmのdo文の前に同様のやり方で制御文を入れて並列化するのは正しくないことに注意してください。これは，異なるスレッドがw2aの同じ要素をアクセスして更新する可能性が出てくるからです。

二電子積分を格納している二次元配列 v(pq,rs)については少し説明が必要です。ここでp,q,r,sは軌道全体に対する添字を表し,最初の1,2,..,NOCCが占有軌道を,そしてNOCC+1,NOCC+2,..,NOCC+NVIRが空軌道を指すとします。ところで,Mullikenの二電子積分$(pq|rs)$は,次のような添字に関する対称性:$(pq|rs)=(qp|rs)=(pq|sr)=(qp|sr)$,$(pq|rs)=(rs|pq)$を持っています。前者の性質を利用し,$p \geq q$および$r \geq s$の組(要するに下三角行列の部分)を,一次元化した添字pqとrsを用いて,二次元配列v(pq,rs)に格納することにすれば,記憶容量を1/4に節約することができます。7,11-13行目は,このルールに基づいて二電子積分vの添字を計算している部分です。配列iaとidxにはそれぞれ,ia(i)=i*(i-1)/2,idx(p,q)=ia(max(p,q))+ min(p,q)が格納されています。$p \geq q$の場合はpq=ia(p)+qで,また,pとqの大小関係が不定の場合はpq=idx(p,q)によって配列要素のアドレスを計算しています。

4．2　ワーク配列を使うプログラム(ARRAY.f)

さて,前のプログラム(SIMPLE.f)はコンパクトで見た目にも単純でいいのですが,配列w2aに書き込むループがmi,ma,mj,mbの順番に入れ子になっていて,メモリアクセスの順番が連続的ではありませんでした。FORTRANの場合,配列の最初の添字が最内側のループ変数になるようにして,2番目,3番目,…の添字を順に外側にすれば,効率の良いメモリアクセスになります。一般にコンピュータでは,

> **今参照している場所から離れた記憶要素をアクセスするのは時間がかかる**

という性質があります。これは,コンピュータは演算装置の下にレジスタ,キャッシュ,メモリ,ディスクといった記憶装置がぶら下がっていて階層構造を作っており,下の階層に行くほどデータの入出力に時間がかかり,それぞれの階層に入りきらない部分はその下の階層に溢れてしまうためです。さらに並列コンピュータの場合は,これらの他にネットワークの階層が付け加わります。プログラムを書く場合は,コンピュータが階層構造の塊であることを常に念頭に置く必要があります。

それでは速いプログラムを書くにはどうしたら良いでしょうか?そのためにはまず,階層構造を越えるようなデータのやり取りを極力しないことです。しかし大きなデータを扱う時は,階層構造を越えるようなことが必ず起こります。その場合でもコンピュータにたいへん都合の良い仕掛けがあって,階層構造の間のギャップを隠蔽するための何らかの緩衝機構(バッファ)が用意されているものです。速いプログラムは,そのような緩衝機構を最大限利用し,階層構造の間のやり取りが実行速度のボトルネックにならないように工夫されています。そのために必要なことは何かと言うと,それは実にとても簡単なことで,

> **同じ要素,または近くにある記憶要素を可能な限り何度も再利用すること**

というたった一つのことだけです。

さて,SIMPLE.fを次のように書き直します。

プログラム例２： ARRAY.f

```
 1      dimension wk1(NVIR,NOCC),wk2(NVIR,NOCC)
 2
 3      do mm=1,NOCC
 4        do me=1,NVIR
 5 !$OMP PARALLEL DO DEFAULT(PRIVATE),SHARED(ia,idx,t1,t2s,t2a,w2a,v,
 6 !$OMP& wk1,wk2,me,mm)
 7          do mi=1,NOCC
 8            do ma=1,NVIR
 9              mea=idx(ma+NOCC,me+NOCC)
10              sum1=0d0
11              sum2=0d0
12              do mf=1,NVIR
13                mem=ia(me+NOCC)+mm
14                mfm=ia(mf+NOCC)+mm
15                mfa=idx(mf+NOCC,ma+NOCC)
16                sum1=sum1+v(mfa,mem)*t1(mf,mi)
17                sum2=sum2+v(mfm,mea)*t1(mf,mi)
18              enddo
19              wk1(ma,mi)=sum1
20              wk2(ma,mi)=sum2
21            enddo
22          enddo
23 !$OMP PARALLEL DO DEFAULT(PRIVATE),SHARED(ia,idx,t1,t2s,t2a,w2a,v,
24 !$OMP& wk1,wk2,me,mm)
25          do mi=1,NOCC
26            do mj=1,NOCC
27              do ma=1,NVIR
28                do mb=1,NVIR
29                  sum=wk1(mb,mj)*(t2s(me,ma,mm,mi)+t2a(me,ma,mm,mi))
30     &              -wk2(mb,mj)*t2a(me,ma,mm,mi)
31     &              -wk2(ma,mj)*t2a(mb,me,mm,mi)
32                  w2a(mb,ma,mj,mi)=w2a(mb,ma,mj,mi)+2.0d0*sum
33                enddo
34              enddo
35            enddo
36          enddo
37        enddo
38      enddo
```

このプログラムは，SIMPLE.f の一時変数 sum1,sum2 の代わりに二次元配列 wk1,wk2 を使っていて，mi に関するループを二つに分割して後半のループの順番が w2a 配列のメモリ配置の順番と一致するようにしたものです。ループを二つに分割したことに伴って，OpenMP の指示制御文も一つ増えています。

このプログラムにはもう一つ，SIMPLE.f と異なる点があります。それは 29-31 行目の t2s,t2a の添字の並び方です。SIMPLE.f では添字 mm の位置が最後であったのに対して，上のプログラム(ARRAY.f)では３番目になっていて，その他の添字の順番も入れ替わっています。これは，$t_{ij}^{ab}, t_{\bar{i}\bar{j}}^{a\bar{b}}$ が添字 i,j と a,b の同時の交換に関して対称であるというカップルドクラスター係数の性質を利用して，添字の順番を入れ替えたものです。このプログラムの後半部分では，mi の添字でループを並列化しているので，共有している配列に複数の**スレッド**（並列動作の単位）から同時にアクセスが起きた時に，参照位置が互いに近くなって競合が起きることがないように配慮する必要があります。ここでは t2s,t2a 配列の添字を入れ替えて mi を最後に持ってくることによって，そのようなアクセスの競合が起きることを抑制しています。

共有メモリ型の並列コンピュータではメモリアクセスの競合に注意が必要

４．３　ブロッキング手法を使うプログラム(BLOCK.f)

前のプログラム(ARRAY.f)では，w2a に対するアクセスは理想的になったのですが，t2s,t2a および v に対するアクセスは連続的ではありません。また，ループの全体の外側には mm と me のループがあって，w2a 配列全体を NOCC*NVIR の回数だけスキャン（舐めるともいう）していて，**「同じ要素，または近くにある記憶要素を可能な限り何度も再利用すること」**という原則に反しています。このような時に使われるのが，**ブロッキング**（または**ブロック化**）という手法です。ここでは，mm と me のループをブロック化した例を示します。

プログラム例３：　BLOCK.f

```
 1      parameter(NBLK=255)
 2      dimension wk1(NBLK,NVIR,NOCC),wk2(NBLK,NVIR,NOCC)
 3
 4      nme=0
 5      do mm=1,NOCC
 6        do me=1,NVIR
 7          nme=nme+1
 8          mne(nme)=me
 9          mnm(nme)=mm
10        enddo
11      enddo
```

```
12
13      nme_total=nme
14      nme_blk=(nme_total-1)/NBLK+1
15
16      do iblk=1,nme_blk
17        nme_start=NBLK*(iblk-1)+1
18        nme_end=min(nme_start+NBLK-1,nme_total)
19
20 !$OMP PARALLEL DO DEFAULT(PRIVATE),SHARED(ia,idx,t1,t2s,t2a,w2a,v,
21 !$OMP& wk1,wk2,mne,mnm,nme_start,nme_end)
22        do mi=1,NOCC
23          do ma=1,NVIR
24            nme=0
25            do nme_ptr=nme_start,nme_end
26              nme=nme+1
27              mm=mnm(nme_ptr)
28              me=mne(nme_ptr)
29              mea=idx(ma+NOCC,me+NOCC)
30              sum1=0d0
31              sum2=0d0
32              do mf=1,NVIR
33                mem=ia(me+NOCC)+mm
34                mfm=ia(mf+NOCC)+mm
35                mfa=idx(mf+NOCC,ma+NOCC)
36                sum1=sum1+v(mfa,mem)*t1(mf,mi)
37                sum2=sum2+v(mfk,mea)*t1(mf,mi)
38              enddo
39              wk1(nme,ma,mi)=sum1
40              wk2(nme,ma,mi)=sum2
41            enddo
42          enddo
43        enddo
44
45 !$OMP PARALLEL DO DEFAULT(PRIVATE),SHARED(ia,idx,t1,t2s,t2a,w2a,v,
46 !$OMP& wk1,wk2,mne,mnm,nme_start,nme_end)
47        do mi=1,NOCC
48          do mj=1,NOCC
49            do ma=1,NVIR
50              do mb=1,NVIR
51                nme=0
```

```
52              sum=0d0
53              do nme_ptr=nme_start,nme_end
54                nme=nme+1
55                mm=mnm(nme_ptr)
56                me=mne(nme_ptr)
57                sum=sum+wk1(nme,mb,mj)*(t2s(me,ma,mm,mi)
58     &                              +t2a(me,ma,mm,mi))
59     &                -wk2(nme,mb,mj)*t2a(me,ma,mm,mi)
60     &                -wk2(nme,ma,mj)*t2a(mb,me,mm,mi)
61              enddo
62              w2a(mb,ma,mj,mi)=w2a(mb,ma,mj,mi)+2.0d0*sum
63            enddo
64          enddo
65        enddo
66      enddo
67    enddo
```

このプログラムでは，mm,me の組を一次元化して，それを NBLK の単位のかたまりに分割して処理しています。（この例では NBLK は 255 に固定しています。）プログラムの最初で me と mm の値をテーブル配列 mne,mnm にストアしています。こうすることで，今処理している me と mm の値がテーブルを引けばすぐにわかるようになっています。

このようなプログラムの書き換えによって mm,me のループを前の例よりも内側に持ってくることができます。この利点は三つあります。一つは w2a 全体をスキャンする回数を減らせること。もう一つはスレッドの起動回数を少なくできること。そしてもう一つは mm,me の組をブロックとしてループの内側で扱うことで，**「メモリアクセスの局所化と同じ要素の再利用」**が容易になることです。これは，このプログラムの後半部分を BLAS ライブラリなどの線形計算パッケージに置き換えることで簡単に取り入れることができます。それを次節に示します。

４．４　ブロッキングとライブラリを併用するプログラム(BLAS.f,MATMPP.f)

前節に書いた通り，BLOCK.f の後半部分は BLAS ライブラリの dgemm 等の行列積サブルーチンを使って書き換えることができます。dgemm 等の最適化された行列積のサブルーチンでは，ブロッキングの手法や**ループアンローリング**といった手法を駆使して，一度アクセスした配列要素が出来る限り再利用されるようなアルゴリズムで計算を行なうため，場合によっては，ほぼ理論ピーク性能に近い速度を出すことができます。（行列積計算の高速化手法については，C や FORTRAN によるコードの事例解説[15,16]があります。）ちなみに，単純なベクトル a と b のスカラー積：

```
sum=0d0; do i=1,N; sum=sum+a(i)*b(i); enddo
```

の計算を単一プロセッサで行なう場合，スピードアップの意味で BLAS ライブラリなどを呼び出す意味は全くありません。これは，上のような演算では a(i)と b(i)の各要素は計算中に一度

しか参照されず，メモリのバンド幅だけで処理速度が決まるため，工夫の余地がないからです。

以下では BLAS の dgemm を用いて BLOCK.f を書き換えた例について，追加配列の宣言部と後半（BLOCK.f の 45 行目以降）の置き換え部分だけを示します。

プログラム例４： BLAS.f（一部）

```
                        ...
      dimension wk3(NVIR,NBLK),wk4(NVIR,NBLK),wk5(NVIR,NBLK)
      dimension wk6(NVIR,NVIR),wk7(NVIR,NVIR),wk8(NVIR,NVIR)
                        ...
 1 !$OMP PARALLEL DO DEFAULT(PRIVATE),SHARED(ia,idx,t2s,t2a,w2a,v,
 2 !$OMP& wk1,wk2,mne,mnm,nme_start,nme_end)
 3       do mi=1,NOCC
 4         nme=0
 5         do nme_ptr=nme_start,nme_end
 6           mm=mnm(nme_ptr)
 7           me=mne(nme_ptr)
 8           nme=nme+1
 9           do ma=1,NVIR
10             wk3(ma,nme)=t2s(me,ma,mm,mi)+t2a(me,ma,mm,mi)
11             wk4(ma,nme)=t2a(me,ma,mm,mi)
12             wk5(ma,nme)=t2a(me,ma,mm,mi)
13           enddo
14         enddo
15         do mj=1,NOCC
16           call dgemm('N','N',NVIR,NVIR,nke,1d0,wk3,NVIR,wk1(1,1,mj),
17     &       NBLK,0d0,wk6,NVIR)
18           call dgemm('N','N',NVIR,NVIR,nke,1d0,wk4,NVIR,wk2(1,1,mj),
19     &       NBLK,0d0,wk7,NVIR)
20           call dgemm('N','N',NVIR,NVIR,nke,1d0,wk5,NVIR,wk2(1,1,mj),
21     &       NBLK,0d0,wk8,NVIR)
22           do ma=1,NVIR
23             do mb=1,NVIR
24               w2a(mb,ma,mj,mi)=w2a(mb,ma,mj,mi)
25     &                         +2.0d0*(wk6(ma,mb)-wk7(ma,mb)
26     &                         -wk8(mb,ma))
27             enddo
28           enddo
29         enddo
30       enddo
```

このプログラムでは，5 行目の nme_ptr のループが BLOCK.f の 53 行目のループに対応しています。t2s,t2a の参照部分は，mi,nme_ptr,ma の 3 重ループの内側になっており，BLOCK.f よりもはるかに少ない参照回数で済むようになったことがこの書き換えの大きなメリットです。

日立の SR11000 では，BLAS ライブラリのほかに MATRIX/MPP という日立製の行列計算ライブラリが用意されています。BLAS の dgemm に対応するルーチンは hdmffm です。インタフェースが dgemm と異なっており，転置行列の積を計算できないという欠点がありますが，上のプログラムはほとんど変更することなく，hdmffm を使ったバージョン(MATMPP.f，リストは省略)にも書き換えることができます。

次節ではこれまでに挙げた５つのプログラムの性能を比較することにします。

４．５　実行テスト結果

プログラムの実行テストは，産業技術総合研究所の SR11000 のほか，IBM p5-560Q，Apple Power Mac G5 Quad を用いました。各コンピュータの性能諸元と使用 OS，コンパイラと使用ライブラリを表１に示します。Power Mac G5 では，用いた IBM XL Fortran for Linux に BLAS ライブラリのサブセット（dgemm を含む）が付属していたので，それを利用しました。

	日立 SR11000 （モデル H1）	IBM p5-560Q	Apple Power Mac G5 Quad
CPU memory	Power4+ 1.7GHz 16proc，64GB(1 node)	Power5+ 1.5GHz 8proc，32GB	PowerPC 970 2.5GHz 4proc，16GB
理論性能	108.8 GFLOPS	48.0 GFLOPS	40.0 GFLOPS
OS	AIX 5L	AIX 5L	Fedora Core 5 (PPC 版)
コンパイラ & ライブラリ	Optimizing FORTRAN90 V01-05-/B MATRIX/MPP V01-04	IBM XL Fortran V9.1 ESSL V4.2	IBM XL Fortran V10.1 for Linux （ライブラリ付属）

表 1:　テストに使用したコンピュータの性能緒元

テスト計算の対象としては，一次元に並んだ Be 原子を選びました。基底関数は最小基底(STO-3G)を用いています。計算規模が増加した場合の性能を評価するために，k 点の数を 15 と 31 に変えた二通りの計算を行ないました。軌道の数:(NOCC,NVIR)に換算すると，それぞれ(15,45),(31,95)となり，後者の場合の計算量は前者のほぼ 83 倍になります。

テストは 4.1 から 4.4 節までに取り上げた５つのコード（SIMPLE，ARRAY，BLOCK，BLAS，MATMPP)を実際のカップルドクラスターのプログラムに組み込み，カップルドクラスター方程式を解く収束計算を 10 回繰り返すことで行ない，各々のコードの実行に要した時間を測定して積算しました。時間の測定は，SR11000 では xclock，p5-560Q と Power Mac G5 では IBM XL Fortran の system_clock サブルーチンを使いました。今回取り上げた式(19)と(20)は演算量があらかじめ決まっていますので，演算量を経過時間で割り算して GFLOPS 値を算出して比較しています。計算に要した時間は k 点が 15 点の場合で 0.7-17 秒，31 点の場合で 53-7000 秒程度でした。

なお，コンパイルオプションは，SR11000 では xf90 -Os -cyclic -64 -omp -save -lmatmpp_sc

-llapack_sc -lblas_sc -lf90c を用い，また，-omp の代わりに-noomp オプションを使って，OpenMP を利用せずに自動並列化機能を使った場合についても調べました。IBM XL Fortran では xlf_r -O3 -w -qsclk=micro -qtune -qhot -qcache -qunroll=yes -qextname -qsmp=omp -q64 を用い，p5-560Q の場合は-lessl，G5 の場合は-qarch=ppc970 を付け加えています。

それでは，k 点を変えた二つのケースに関する実行結果を以下（表２，３）に示します。

	SR11000 OpenMP	SR11000 自動並列化	p5-560Q	Power Mac G5 Quad
SIMPLE	7.6	7.7	4.3	1.5
ARRAY	21.8	23.1	12.5	2.6
BLOCK	14.0	14.2	6.7	3.2
BLAS	21.7	1.9	22.7	3.5
MATMPP	38.6	5.1	-	-

表２：　k 点が 15 点の場合の各プログラムの実行速度（単位は GFLOPS）

	SR11000 OpenMP	SR11000 自動並列化	p5-560Q	Power Mac G5 Quad
SIMPLE	1.4	1.3	1.7	0.3
ARRAY	10.8	5.2	14.1	2.4
BLOCK	12.9	13.1	5.6	3.0
BLAS	33.2	4.1	24.2	4.5
MATMPP	36.9	5.2	-	-

表３：　k 点が 31 点の場合の各プログラムの実行速度（単位は GFLOPS）

まず SR11000 で OpenMP を利用した場合の結果を見ると，SIMPLE の性能が非常に悪いことがわかります。システムのサイズを大きくするとその傾向は特に顕著になります。また，ARRAY の性能は，システムのサイズが小さいときはまあまあですが，サイズを大きくすると性能は半分近くになります。BLOCK，MATMPP ではシステムのサイズを大きくしてもさほど性能が変わりません。これは SIMPLE や ARRAY の性能がキャッシュの大きさに強く依存しており，扱うデータが大きくなるとキャッシュに入りきらずに性能が低下するのに対して，BLOCK や MATMPP ではブロッキング手法を使っているため，そのようなことがないのが理由です。MATMPP は今回テストした中で最も性能がよく，SR11000 の理論性能の 30%以上を出しています。BLAS の方は MATMPP に比べて性能が落ちています。詳しく調べてみたところ，SR11000 の BLAS ライブラリの dgemm は MATRIX/MPP ライブラリの hdmffm に比べて性能が悪く，特に計算サイズが小さい時の性能が良くないことがわかりました。SR11000 では BLAS の利用を避けた方が賢明のようです。（現在 BLAS ライブラリが業界標準になっていることを考えると，これはちょっと困ったことですが。）

SR11000 で自動並列化を使った場合を見ると，SIMPLE，ARRAY，BLOCK では OpenMP を使った場合と傾向が似ています。特に，計算サイズが小さい場合は，OpenMP を使った場合の性能と

大差がありません。ところがBLAS，MATMPPでは，自動並列化を使うと大きく性能が低下しています。この理由は良くわかりませんが，自動並列化はまだ万能ではないということが言えそうです。

p5-560Qの場合でもSIMPLEの性能が悪い点はSR11000と同じです。BLASの性能は非常に良好で，p5-560Qの理論性能のほぼ半分出ています。ARRAYもまあまあの性能を出していて，BLASの場合もそうですが，サイズを大きくした時に速度が向上している点が目を引きます。一方，BLOCKの性能はSR11000と大きく異なり，全く良くありません。

最後にPower Mac G5の場合です。系のサイズを大きくした時にSIMPLEの性能が大きく低下している点，ARRAYでは性能が向上している点はp5-560Qと共通です。ところがBLOCKでは，p5-560QとおなじIBMコンパイラを使っているにもかかわらず，ARRAYよりも良い性能が出ています。また，BLASの性能は他の三つのプログラムよりも良いのですが，他の機種で見られたほどの性能の向上はありませんでした。

さて，これらの結果を俯瞰すると，今回扱った計算はプログラムの書き方で大きく速度が異なること，特にブロック化と行列計算ライブラリを併用する方法が非常に有効であること，また，プログラムの書き換えによる効果の度合いは，使用するコンピュータやコンパイラ，系のサイズに強く依存するということがお分かりいただけると思います。ブロッキング手法等を持ち込むとプログラムは非常に複雑になり，労力もそれなりにかかりますので，どの程度のチューニングが必要かの判断は，プログラムの目的を十分考慮して考える必要があるでしょう。ここでは，プログラムが実際に何GFLOPS出ているかを知ることが，チューニングの必要性を客観的かつ適切に判断する上で非常に大切である，ということを強調しておきたいと思います。

それにしても，現在のコンピュータ（スパコンも含む）は，洗練された道具と言うには，いまだほど遠いものだと言えるでしょう。性能を引き出すためには，かなり細かいことをコンピュータに指示しなければなりません。プロセッサがベクトル型から現在のプロセッサに変遷してから，その傾向はさらに顕著になっている気がしています。スーパーコンピューティングがオリンピックならば（実際そういう側面もあると思いますが），性能を引き出すための細かい工夫は意味のあることでしょうが，できればそういったことはしないで済むに越したことはありません。もっとスマートなコンピュータの登場を望みたいものです。

なお，ライブラリを利用してプログラムを書く場合には，4.4のBLAS.fのように，特定のライブラリのサブルーチン呼び出しをコードの中に埋め込む方法を取らず，間にプログラマーが決めた別の名前のサブルーチン（ラッパーと言う）を入れて間接的に呼び出すような方法を取ることをお勧めします。この方法を使えば，ライブラリを変更しても（例えばBLAS->MATRIX/MPP），ソースコードの変更を最小限で済ますことができます。

５．おわりに

本稿では，カップルドクラスター法と高分子の計算への適用について概要を紹介し，カップルドクラスター係数の積和計算を取り上げて，そのプログラミング手法について紹介しました。このような積和計算は，カップルドクラスター法のような多体問題のダイアグラム計算だけではなく，様々な数値計算のアプリケーションに出てくると思います。拙稿で紹介した事例が，これを読まれた方の参考になれば幸いです。なお本稿ではカップルドクラスター法を高分子に応用した場合の成果については触れませんでした。それについては文献[4,13]をご覧ください。

今，次世代のスパコンでは数万以上のプロセッサが並列に動作することが想定されています。カップルドクラスター法の計算をそのようなシステムで効率よく行なうには，いろいろな問題を解決する必要があります。例えば，カップルドクラスター法で扱うカップルドクラスター係数や二電子積分は，系のサイズの４乗のオーダーで大きくなりますので，１ノード内のメモリに収まらない場合にどうするかを考慮しなければなりません。その場合，実際に１ノードが持つメモリ量によって取る戦略が変わってきます。またこのほかに，数万以上のプロセッサで並列化するためには，どういった並列要素を利用するかということも考える必要が出てくるでしょう。こういった問題は他の多くの電子状態計算法に共通のもので，今後の課題であると言えます。

なお，本稿の成果の一部は，日本学術振興会の科学研究費補助金特定領域研究「次世代共役ポリマーの超階層制御と革新機能」の援助を受けて得られました。また長嶋雲兵氏からは，本稿について有難いコメントを頂戴しました。この場を借りて感謝したいと思います。

参 考 文 献

[1] A・ザボ／N・S・オストランド：大野公男／阪井健男／望月祐志訳『新しい量子化学（上・下)』，東京大学出版会，1988.

[2] R・G・パール／W・ヤング：狩野覚／関元／吉田元二訳『原子・分子の密度汎関数法』，シュプリンガー・フェアラーク東京，1996.

[3] B.L. Hammond, W.A. Lester, P.J. Reynolds, *Monte Carlo Methods in Ab initio Quantum Chemistry* (World Scientific, 1994).

[4] H. Katagiri, J. Chem. Phys. **122**, 224901 (2005).

[5]F. Coester, Nucl. Phys. **7**, 421 (1957); F. Coester and H. Kümmel, Nucl. Phys. **17**, 477 (1960).

[6] J. Cizek, J. Chem. Phys. **45**, 4256 (1966); J. Cizek, Adv. Chem. Phys. **14**, 35(1969).

[7] http://www.emsl.pnl.gov/forms/basisform.html

[8] M･T･バチェラー:田崎晴明訳『ベーテ仮設の７５年』パリティ,Vol.22, No.09, 丸善, (2007).

[9] R.J. Bartlett, in *Modern Electronic Structure Theory Part II*, ed. D.R. Yarkony, (World Scientific, 1995), pp.1047-1131.

[10] C. Pisani, R. Dovesi, and C. Roetti, *Hartree-Fock Ab Initio Treatment of Crystalline Systems*, Lecture Notes in Chemistry Vol. 48 (Springer, Heidelberg, 1988).

[11] S. Suhai, Phys. Rev. B **27**, 3506 (1983).

[12] J.-Q. Sun and R. J. Bartlett, J. Chem. Phys. **104**, 8553 (1996).

[13] S. Hirata, R. Podeszwa, M. Tobita, R.J. Bartlett, J. Chem. Phys. **120**, 2581 (2004).

[14] http://www.sgi.co.jp/origin/ODP/documents/programming/openmp/index.html

[15] 前野，太田：情報処理学会研究報告(93-HPC-49)，Vol.93, No. 89, pp.1-8 (1993).

[16] 山本，小畑，村上，片桐，秦野：情報処理学会研究報告(95-HPC-55)，Vol.95, No.55, pp.57-64 (1995)；山本，秦野：名古屋大学大型計算機センターニュース Vol.26, No.3, pp.182-193 (1995).