JAERI-Data/Code
96-007

カー・パリネロ法のベクトル並列化

1996年3月

田中宏志

日本原子力研究所
Japan Atomic Energy Research Institute

本レポートは，日本原子力研究所が不定期に公刊している研究報告書です。
入手の問合わせは，日本原子力研究所技術情報部情報資料課（〒319-11 茨城県那珂郡東海村）あて，お申し越しください。なお，このほかに財団法人原子力弘済会資料センター（〒319-11 茨城県那珂郡東海村日本原子力研究所内）で複写による実費頒布をおこなっております。

This report is issued irregularly.
Inquiries about availability of the reports should be addressed to Information Division, Department of Technical Information, Japan Atomic Energy Research Institute, Tokai-mura, Naka-gun, Ibaraki-ken 319-11, Japan.

編集兼発行　日本原子力研究所
印　　刷　いばらき印刷㈱

JAERI-Data/Code 96-007

カー・パリネロ法のベクトル並列化

日本原子力研究所計算科学技術推進センター
田中　宏志

(1996年2月7日受理)

カー・パリネロ法に基づく第一原理分子動力学プログラムCamp-Atamiを、スーパーコンピュータVPP500用にベクトル並列化した。このプログラムは、電子構造計算で良く用いられる共通の計算手法を幾つか含んでいる。並列化ではできるだけ大粒度の並列化を試み、逆格子点毎の計算とエネルギーバンド毎の計算を並列化することで全体の9割程度を並列化した。並列化によるスピードアップは8プロセッサーまでほぼ一様に増加し、8プロセッサーで4.4倍であった。また、扱えるデータを2基底に制限することで高速な並列版3次元高速フーリエ変換プログラムを新たに作成した。その結果、複数のプロセッサーで実行した時には、提供されているライブラリーよりもはるかに速い処理能力を示すことができた。

日本原子力研究所：〒153　東京都目黒区中目黒2－2－54

JAERI-Data/Code 96-007

Vector-parallel Processing of Car-Parrinello Method

Hiroshi TANAKA

Center for Promotion of Computational Science and Engineering
Japan Atomic Energy Research Institute
Nakameguro, Meguro-ku, Tokyo

(Received February 7, 1996)

First principles molecular dynamics program 'Camp Atami' is parallelized for VPP500, which is a vector parallel type supercomputer. The program is based on Car-Parrinello method, and includes common techniques frequently used in calculations of electronic structures. In order to get large granularity, the parallelization is conducted for each reciprocal lattice point and for each band index, and reached up to about 90% of the total code. Speedup ratio increases almost proportionally up to 4.4 times with the increase of processor number up to 8. A new 3 dimensional fast Fourier transform (FFT) program is made and parallelized. The result is found to be much faster than that provided as a subroutine library by restricting the number of data elements to the power series of 2.

Keywords: Car-Parrinello Method, First Principles MD, VPP500, Vector-parallel, FFT, Speedup Ratio

目　　次

Contents

1. はじめに

カーとパリネロは，物理的な考察に基づき第一原理的分子動力学を飛躍的に高速化するアルゴリズムを開発した [1]．これをカー・パリネロ（CP）法と呼ぶ．これにより，現在の CPU 能力では事実上不可能であった現実的な系での第一原理分子動力学が可能になった．一方，近年スーパーコンピュータは複数の CPU で並列処理を行う並列型が主流になりつつある．並列型コンピュータの利点は，計算時間が短縮されるのと同時に，大きなメモリー空間を使用できることである．従って，カー・パリネロ法を並列化することでより大きな系（単位胞あたりに含む原子の数が大きい系や，表面界面を含む系）において，より高いエネルギー分解能での分子動力学が可能になる．そこでカー・パリネロ法による分子動力学プログラムを並列化することを試みた．

現在，この方法を用いた第一原理分子動力学のプログラムコードは幾つかが公開され，計算物理を職業としている人以外の実験家などでも比較的簡単に利用できるようになってきている．なかでも日本化学計算プログラム交換機構を通して配布されている Camp-Atami[2] は，もともと公開することを前提に書かれており，非常に教科書的に書かれていて，汎用性も高い．研究者が作ったプログラムにありがちな，計算に必要なパラメータ等の指定が作った本人にしか解らないといったこともない．そこで汎用性と読みやすさ，また最終生成物が公開可能であるという観点からこの Camp-Atami を選びこれをベクトル並列化した．

この報告書では，第一原理分子動力学にあまり馴染みのない人でも理解できるように 2 章に簡単な説明をした．この章では第一原理による電子構造計算の基礎となる局所密度汎関数法の説明から始めて，従来の第一原理分子動力学法とカー・パリネロ法の比較，そしてカー・パリネロ法の実際の定式化をできるだけ本報告書の中だけで理解できるようにした．

第 3 章では Camp-Atami のプログラムの流れをできるだけ詳しく説明した．それぞれの副プログラムが第 2 章で説明した定式のどの部分を計算しているのか，あるいはその計算結果はどの変数に保存されているのか等を示した．これは，これからこのプログラムを解読し，拡張改良したり並列化したりする人の手助けとするためである．

第 4 章はベクトル並列化の観点からデータの流れを中心に重要な副プログラムについて実際のプログラムコードの構造を示すとともに，どのように並列化を行ったかを示した．高速フーリエ変換の並列化についての部分は，このプログラムに限らず一般的に使えるものである．

第 5 章は今回行ったベクトル並列化による性能向上の評価である．最後に第 6 章は，全体のまとめと，今後 Camp-Atami をさらに改良していく上で重要と思われるポイントを並列化の観点から示した．

プログラムの並列化にあたっては，PE（Procesor Element）間のデータ転送速度が重要である．そこで，並列化の過程で明らかになった VPP でのデータ転送の特性を付録 1 に示した．また，付録 2 にはバンク競合と並列化の問題について触れた．付録 3 には，並列化したプログラムを実際にコンパイルして実行する手順を示した．

今回並列化を試みたCamp-Atamiは，非常に教科書的に書かれていてこれから分子動力学を使おうとしている人にも，またこれを改良して自分の目的にあったプログラムを作成しようという人にも良くできている．プログラムは少し長いが，大変見易く書かれているので，この報告書を読んで少しでも多くの人がこのプログラムを解読し改良する手助けになれば幸いである．

2. 第一原理分子動力学

2.1 局所密度汎関数法

一般に第一原理による電子構造計算といった場合，局所密度近似を用いた密度汎関数法による計算をさす事が多く，ここで並列化を行った Camp-Atami もこの局所密度汎関数法を用いている．密度汎関数理論によれば [3]，多電子系の基底状態は 1 電子密度$\rho(\boldsymbol{r})$ で一義的に決り，その基底状態エネルギーを

$$E[\rho(\boldsymbol{r})] = F[\rho(\boldsymbol{r})] + \int d\boldsymbol{r} v(\boldsymbol{r})\rho(\boldsymbol{r}), \tag{2.1}$$

とすると，$E[\rho(\boldsymbol{r})]$ は正しい基底状態の$\rho(\boldsymbol{r})$ に対して最小値をとることが Hohenberg と Kohn により証明されている．ただし $v(\boldsymbol{r})$ は外場によるポテンシャルである．ここで，$F[\rho(\boldsymbol{r})]$ を

$$F[\rho(\boldsymbol{r})] = T_s[\rho(\boldsymbol{r})] + \frac{1}{2}\int d\boldsymbol{r} d\boldsymbol{r}' \frac{\rho(\boldsymbol{r})\rho(\boldsymbol{r}')}{|\boldsymbol{r}-\boldsymbol{r}'|} + E_{xc}[\rho(\boldsymbol{r})] \tag{2.2}$$

と書くことにする．ただし $T_s[\rho(\boldsymbol{r})]$ は相互作用のない時の運動エネルギーであり，第 2 項はクーロンエネルギー，$E_{xc}[\rho(\boldsymbol{r})]$ は交換相関エネルギーである．電子密度が十分空間的にゆっくりと変化するとすれば，交換相関エネルギーは，

$$E_{xc}[\rho(\boldsymbol{r})] \simeq \int d\boldsymbol{r} \rho(\boldsymbol{r}) \epsilon_{xc}(\rho(\boldsymbol{r})) \tag{2.3}$$

と書ける．これは，交換相関エネルギー密度ϵ_{xc} が局所電子密度の関数で与えられるとした近似で局所密度近似と呼ぶ．規格直交条件を満たす一電子波動関数を$\psi_i(\boldsymbol{r})$ とすると，電子密度は

$$\rho(\boldsymbol{r}) = \sum_i n_i |\psi_i(\boldsymbol{r})|^2 \tag{2.4}$$

で表わされる．ここで，iは固有状態のインデックスを示し，n_iは i 番目の固有状態の電子占有率を示す．式 (2.1) を$\psi_i(\boldsymbol{r})$ に関して変分すると 1 電子シュレディンガー方程式として次式が与えられる．

$$\left[-\frac{\hbar^2}{2m}\nabla^2 + e^2\int d\boldsymbol{r}' \frac{\rho(\boldsymbol{r}')}{|\boldsymbol{r}-\boldsymbol{r}'|} + \frac{\delta E_{xc}[\rho(\boldsymbol{r})]}{\delta\rho} + v(\boldsymbol{r})\right]\psi_i(\boldsymbol{r}) = \epsilon_i\psi_i(\boldsymbol{r}), \tag{2.5}$$

ここで，ϵ_iは i 番目の固有状態の固有エネルギーである．これを Kohn-Scham 方程式と呼ぶ [4].

2.2 カー・パリネロ法

第一原理分子動力学と呼ばれる多くの方法は前節で述べた局所密度汎関数近似に基づいて電子状態の計算を行っている．Fig. 2.1にカー・パリネロ法以前に行われていた一般的な第一原理分子動力学計算法の流れ図を示す．まず，原子の座標$\boldsymbol{R}_n$を決定し，この原子配置のもとに Kohn-Sham 方程式 (2.5) を自己無撞着に解く．具体的には Kohn-Sham 方程式により得られるハミルトニアン

を対角化し固有エネルギーϵ_iと波動関数$\psi_i(\boldsymbol{r}, \boldsymbol{R}_n)$を求め，新しい電子密度を$\rho(\boldsymbol{r}) = \sum_i |\psi_i(\boldsymbol{r})|^2$により求める．この電子密度から新しいハミルトニアンを作り，これを対角化して新しい固有エネルギーと波動関数を求める．この手順を自己無撞着になるまで繰り返し，与えられた原子配置に対して全エネルギーを最少にする波動関数を求める（波動関数をクエンチする）．次に各原子に働く力$\boldsymbol{f}_n$を，$\boldsymbol{f}_n = -\frac{\partial}{\partial \boldsymbol{R}_n} E[\{\boldsymbol{R}_n\}, \{\psi_i\}]$から計算し，求まった$\boldsymbol{f}_n$から新しい原子の座標を決める．新しい原子配置のもとに再び Kohn-Sham 方程式を自己無撞着に解く．以上の手順を必要な回数だけ繰り返す．

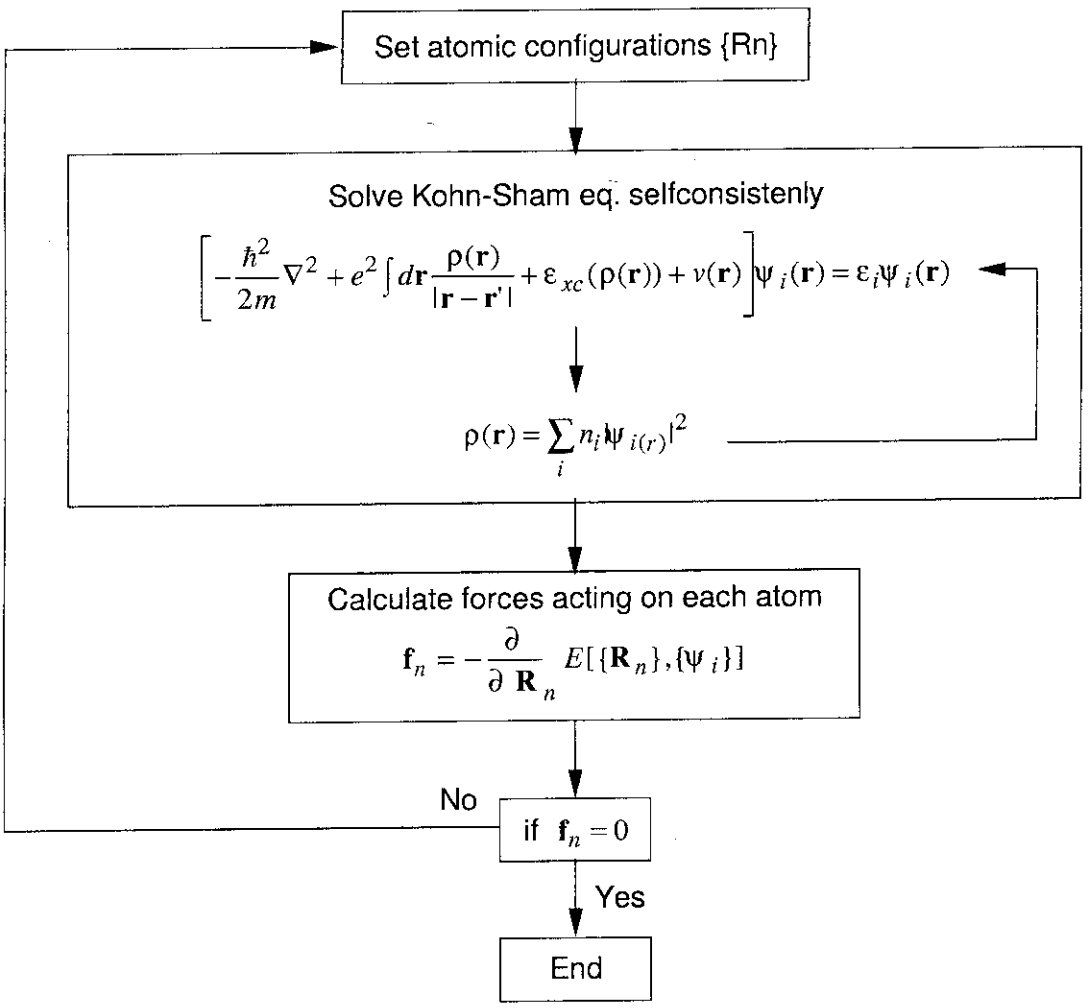

Fig. 2.1　従来法による第一原理分子動力学

この手順の中で，2番目のステップであるハミルトニアンの対角化の部分が多くの記憶容量と演算処理時間を必要とし，結果として現実的な系での分子動力学を困難にしていた．Car と Parrinello は波動関数に仮想的な運動エネルギーを導入して仮想的なラグランジュアンを用いることで，電子系$\{\psi_i\}$と格子系$\{\boldsymbol{R}_n\}$を独立なパラメターとして扱い，従来法で必要であった電子系のハミル

トニアンを対角化するプロセスを省略する方法を提案した．これをカー・パリネロ法と呼ぶ．この方法で使用されるラグランジュアンは，

$$L = \mu \sum_i \int d\boldsymbol{r} |\dot{\psi}_i(\boldsymbol{r},t)|^2 + \frac{1}{2}\sum_n M_n \dot{\boldsymbol{R}}_n^{\ 2} - E[\{\psi_i\},\{\boldsymbol{R}_n\}] + \sum_{i,j} \Lambda_{ij}(\int d\boldsymbol{r}\psi_i\psi_j - \delta_{ij}) \tag{2.6}$$

である．第1項，第2項はそれぞれ波動関数系と格子系の運動エネルギーで，第3項は電子系の全エネルギーである．最後の項は波動関数 $\{\psi_i\}$ の規格直交条件による拘束であり，Λ_{ij}はラグランジュの未定係数である．式 (2.6) から，$\{\psi_i\}$，$\{\boldsymbol{R}_n\}$ の満たす運動方程式は，

$$\mu\ddot{\psi}_i(\boldsymbol{r},t) = -\frac{\delta E}{\delta\psi_i^*(\boldsymbol{r},t)} + \sum_j \Lambda_{ij}\psi_j(\boldsymbol{r},t), \tag{2.7}$$

$$M_n\ddot{\boldsymbol{R}}_n = -\frac{\partial}{\partial \boldsymbol{R}_n}E. \tag{2.8}$$

で与えられることが判る．式 (2.7) は定常状態で，

$$\frac{\delta E}{\delta\psi_i^*(\boldsymbol{r},t)} = \sum_j \Lambda_{ij}\psi_j(\boldsymbol{r},t) \tag{2.9}$$

となり，適当なユニタリー変換により Kohn-Sham 方程式 (2.5) を満たすことが分かる．従って，運動方程式 (2.7)-(2.8) に従って $\{\psi_i\}$，$\{\boldsymbol{R}_n\}$ を時間発展させたとき $\{\psi_i\}$ は自動的に電子系のハミルトニアンを対角化する形で発展していく事が分かる．つまり，ハミルトニアンを陽に対角化するプロセスを省略して分子動力学を行える．

Fig. 2.2にカー・パリネロ法の概念図を示す．図中の曲面は，原子配置 $\{\boldsymbol{R}_n\}$ と波動関数 $\{\psi_i\}$ の張る位相空間での系の全エネルギーを示す．また図中の実線は様々な原子配置に対して全系のエネルギーが最少になる点を結んだもので Born-Oppenheimer (BO) 面と呼ばれるものである．この図で従来の第一原理分子動力学の手続きを説明すると，まず原子配置を固定してその平面内で全エネルギーが最少になる波動関数（したがって電子配置）をシュレディンガー方程式を解くことで求める．次に各原子に働く力を求め，その力に従って各原子を少し動かし再び新しい原子配置のもとにシュレディンガー方程式を解いて全エネルギーが最少となる波動関数を求める．この一連の手続きをくり返し行うことにより，BO 面にそって全系のエネルギーが最少になる点を探すことになる．一方，カー・パリネロ法では式 (2.7)-(2.8) の運動方程式に従って実際に時間発展を追うことで BO 面の近傍にそって自動的に全エネルギーが最少の点に収束していく．

カー・パリネロ法では，次に説明する擬ポテンシャル法と組み合わせて，波動関数は平面波展開したものを用いることが多い．これはカー・パリネロ法の制限ではないが，平面波を使うと後に述べるように原子に働く力の計算の一部が簡単になるからである．

2.3 擬ポテンシャル法と平面波展開

真のポテンシャルが，内殻の電子により十分遮蔽されていて，なおかつ価電子が s，p 軌道だけから成るような原子では，価電子は深いポテンシャルを感じない．そのため深い真のポテンシャルをより浅い実効的なポテンシャルに置き換えて電子構造を計算する事が出来る．これを擬ポテ

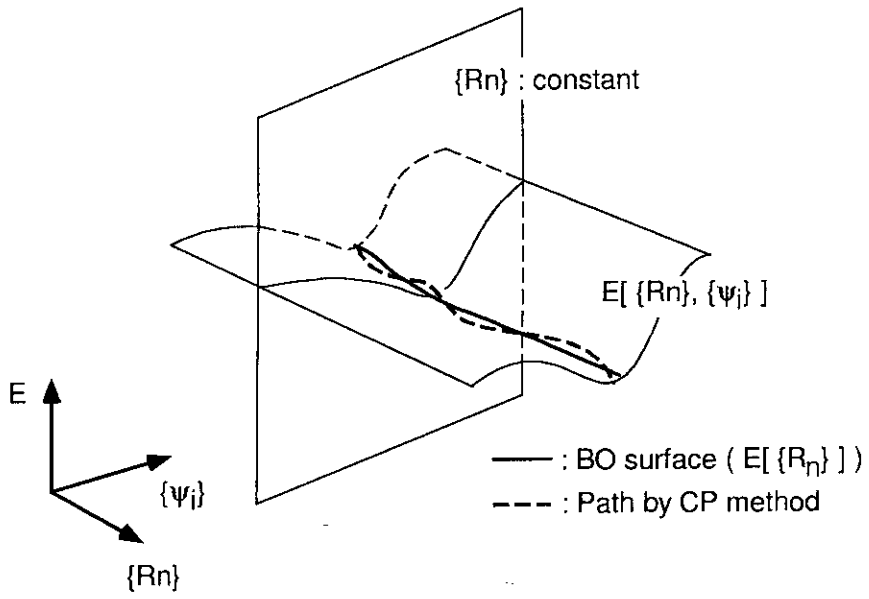

Fig. 2.2 カー・パリネロ法の概念図

ンシャル法と呼ぶ．従来擬ポテンシャルは経験的に求められてきたが，最近は第一原理的に求める事ができ，かついろいろな系に適用が可能な可搬性を持ったものが開発されてきた．その1つが以下に紹介するノルム保存型擬ポテンシャル [5] と呼ばれるものである．ノルム保存型擬ポテンシャルは次の4つの特徴を持つ．

- 真のポテンシャルと擬のポテンシャルに対する価電子の固有値がある原子構造に関して一致し，かつ擬ポテンシャルに対応する擬の波動関数は，動径方向で符号を変えることがない（節を持たない）．
- 真と擬の価電子の電荷密度がコア半径 r_c の外側で一致する．
- 真と擬の波動関数を原点からコア半径 r_c まで積分したものは一致する（ノルム保存型と呼ばれる所以）．
- 真と擬の波動関数の $r = r_c$ での対数微分とエネルギー微分が一致する．

これらの条件を満すように求めた擬ポテンシャルは Bachelet，Hamann と Schlüter[6] により H から Pu までの原子について求められており，角運動量のプロジェクションオペレータを用いて，

$$V_{ps}(\boldsymbol{r}) = \sum_l V_{ps,l}(r)\hat{P}_l \tag{2.10}$$

と表わされている．これは更に，角運動量に依存する部分と依存しない部分に分ける事が出来て，

$$V_{ps}(\boldsymbol{r}) = V_L(\boldsymbol{r}) + \sum_l V_{NL,l}(\boldsymbol{r})\hat{P}_l \tag{2.11}$$

と書く事が出来る．$V_L(\boldsymbol{r})$ と $V_{NL,l}$ については表式が与えられていて，結果だけ示すと，

$$V_L(\boldsymbol{r}) = -\frac{Z_v}{r}\sum_{i=1}^{2} C_i erf(\sqrt{\alpha_i} r) \tag{2.12}$$

$$V_{NL,l}(\boldsymbol{r}) = \sum_{i=1}^{3}(A_i^l + r^2 A_{i+3}^l)\exp(-\alpha_i^l r^2) \tag{2.13}$$

となっている．ここで Z_vは原子番号であり，その他のパラメータは文献 [6] に表で与えられている．

擬ポテンシャルの様に浅いポテンシャルを用いた時は，対応する（擬の）波動関数の空間的な変化がゆるやかであるため少数の平面波で展開する事が出来て，

$$\psi_i(\boldsymbol{r}) = \frac{1}{\sqrt{\Omega}}\sum_{\boldsymbol{G}} C_i^{\boldsymbol{k}+\boldsymbol{G}} \cdot e^{i(\boldsymbol{k}+\boldsymbol{G})\cdot\boldsymbol{r}} \tag{2.14}$$

となる．

2.4 カー・パリネロ法の表式

前節で紹介したノルム保存型擬ポテンシャルを用いた時のカー・パリネロ法の表式を示す．まず，局所密度汎関数近似に基づく全エネルギーは実空間表示で以下の様になる．

$$\begin{aligned} E &= E_{kin} + E_{coul} + E_{xc} + E_{ele-ion} + E_{ion-ion} \\ &= \sum_i \int d\boldsymbol{r}\psi_i^*(\boldsymbol{r})\left(-\frac{1}{2}\nabla^2\right)\psi_i(\boldsymbol{r}) \\ &+ \frac{1}{2}\int\int d\boldsymbol{r}d\boldsymbol{r}'\frac{\rho(\boldsymbol{r})\rho(\boldsymbol{r}')}{|\boldsymbol{r}-\boldsymbol{r}'|} + \int d\boldsymbol{r}\rho(\boldsymbol{r})\epsilon_{xc}(\rho(\boldsymbol{r})) \\ &+ \sum_i \int d\boldsymbol{r}\psi_i^*(\boldsymbol{r})\sum_{\boldsymbol{R}_n,l} V_{ps,l}(\boldsymbol{r}-\boldsymbol{R}_n)\hat{P}_l\psi_i(\boldsymbol{r}) \\ &+ \frac{1}{2}\sum_{n\neq n'}\frac{Z_n Z_{n'}}{|\boldsymbol{R}_n - \boldsymbol{R}_{n'}|}. \end{aligned} \tag{2.15}$$

ここで第 1 項から第 3 項はそれぞれ，電子の運動エネルギー，クーロンエネルギー，交換相関エネルギーである．第 4 項はポテンシャルエネルギーでポテンシャルは前節で紹介した擬ポテンシャルに差し替えられている．最後の項は，イオン同士の静電エネルギーである．

以下では，i 番目のバンドに対応する波動関数が式 (2.14) の様に平面波で展開した時の式 (2.15) の逆格子空間での表式を考える．電子の運動エネルギーの項は，

$$E_{kin} = \frac{1}{2}\sum_i\sum_{\boldsymbol{G}}|\boldsymbol{k}+\boldsymbol{G}|^2 \tag{2.16}$$

で与えられる．ここで i はバンドインデックスである．クーロンポテンシャル $V_{Coul}(\boldsymbol{r})$ を，

$$V_{Coul}(\boldsymbol{r}) = \int d\boldsymbol{r}'\frac{\rho(\boldsymbol{r}')}{|\boldsymbol{r}-\boldsymbol{r}'|}, \tag{2.17}$$

で定義し，これと電荷密度$\rho(\boldsymbol{r})$ のフーリエ展開を

$$\rho(\boldsymbol{r}) = \sum_{\boldsymbol{G}} \rho(\boldsymbol{G}) e^{i\boldsymbol{G}\cdot\boldsymbol{r}},\ V_{Coul}(\boldsymbol{r}) = \sum_{\boldsymbol{G}} V_{Coul}(\boldsymbol{G}) e^{i\boldsymbol{G}\cdot\boldsymbol{r}} \tag{2.18}$$

とすると，そのフーリエ係数は，

$$\rho(\boldsymbol{G}) = \sum_i \sum_{\boldsymbol{G}'} (C_i^{\boldsymbol{G}+\boldsymbol{G}'})^* C_i^{\boldsymbol{G}'},\ V_{Coul}(\boldsymbol{G}) = \frac{4\pi\rho(\boldsymbol{G})}{|\boldsymbol{G}|^2} \tag{2.19}$$

で与えられる．ここで*は複素共役を示す．これらを用いると，クーロンエネルギーは，

$$E_{Coul} = \frac{1}{2}\int\int d\boldsymbol{r}d\boldsymbol{r}' \frac{\rho(\boldsymbol{r})\rho(\boldsymbol{r}')}{|\boldsymbol{r}-\boldsymbol{r}'|} = \frac{1}{2}\int d\boldsymbol{r} V_{Coul}(\boldsymbol{r})\rho(\boldsymbol{r}) = \frac{1}{2}\sum_{\boldsymbol{G}} V_{Coul}(\boldsymbol{G})\rho^*(\boldsymbol{G}) \tag{2.20}$$

で与えられる．同様に$\epsilon_{xc}(\boldsymbol{r})$ を，

$$\epsilon_{xc}(\boldsymbol{r}) = \sum_{\boldsymbol{G}} \epsilon_{xc}(\boldsymbol{G}) e^{i\boldsymbol{G}\cdot\boldsymbol{r}} \tag{2.21}$$

とフーリエ展開すると，交換相関エネルギーは，

$$E_{xc} = \int d\boldsymbol{r}\epsilon_{xc}(\boldsymbol{r})\rho(\boldsymbol{r}) = \sum_{\boldsymbol{G}} \epsilon_{xc}(\boldsymbol{G})\rho^*(\boldsymbol{G}) \tag{2.22}$$

と書ける．次に，擬ポテンシャルによるエネルギーであるが，擬ポテンシャルは 2.3 節で述べたように角運動量に依存しない局所的な部分と依存する非局所的な部分に分解されて，

$$E_{ele-ion} = \sum_i \sum_{\boldsymbol{R}_n} \int d\boldsymbol{r}\psi_i^*(\boldsymbol{r})(V_L(\boldsymbol{r}-\boldsymbol{R}_n) + \sum_l V_{NL,l}(\boldsymbol{r}-\boldsymbol{R}_n))\psi_i(\boldsymbol{r}) \tag{2.23}$$

これを，式 (2.13)-(2.12) と比べると，ポテンシャルの局所的部分からは，

$$\int d\boldsymbol{r} e^{i(\boldsymbol{G}'-\boldsymbol{G})\cdot\boldsymbol{r}} \frac{1}{|\boldsymbol{r}-\boldsymbol{R}_n|} erf(\sqrt{\alpha_j^n}|\boldsymbol{r}-\boldsymbol{R}_n|) \tag{2.24}$$

という形の積分が必要になることが解る．この積分は解析的に行えて負荷もそれ程大きくはない．一方，非局所的な部分では，

$$\int d\boldsymbol{r} e^{-i(\boldsymbol{k}+\boldsymbol{G})\cdot\boldsymbol{r}} \sum_{i=1}^{3} \left[(A_i^l + r^2 A_{i+3}^l)e^{-\alpha_i^l r^2}\right] \hat{P}_l(\boldsymbol{r}) e^{i(\boldsymbol{k}+\boldsymbol{G}')\cdot\boldsymbol{r}}$$

$$= 4\pi(2l+1)P_l(\cos\theta_{\boldsymbol{G}\boldsymbol{G}'}) \int d\boldsymbol{r} \sum_{i=1}^{3} \left[(A_i^l + r^2 A_{i+3}^l)e^{-\alpha_i^l r^2}\right] j_l(|\boldsymbol{k}+\boldsymbol{G}|r) j_l(|\boldsymbol{k}+\boldsymbol{G}'|r) \tag{2.25}$$

という形の積分が必要になる．ここで，$\hat{P}_l$は角運動量のプロジェクションオペレータであり，j_jは球ベッセル関数である．この積分は，平面波の数が n 個とすると $n\times(n+1)/2$ 回の積分を必要とし，系が大きくなると急速に負荷が大きくなる．これを回避するために分離型擬ポテンシャルと呼ばれる方法 [7] も提案されているが，今回のプログラムでは使用されていない．最後のイオン同士の静電エネルギーの項は，詳細は省略するが Ewald の方法 [8] により計算される．

以上をまとめると，局所密度汎関数法と擬ポテンシャル法を用いた逆格子空間での全エネルギーの表式は，

$$
\begin{aligned}
E_{LDF} &= \frac{1}{2}\sum_i\sum_{\boldsymbol{G}}|\boldsymbol{k}+\boldsymbol{G}|^2 \\
&+ \frac{1}{2}\sum_{\boldsymbol{G}}V_{coul}(\boldsymbol{G})\rho^*(\boldsymbol{G})+\sum_{\boldsymbol{G}}\epsilon_{xc}(\boldsymbol{G})\rho^*(\boldsymbol{G}) \\
&+ \sum_i\sum_{\boldsymbol{G}\boldsymbol{G}'}(C_i^{\boldsymbol{G}})^*C_i^{\boldsymbol{G}'}V_{ps}^{\boldsymbol{G}\boldsymbol{G}'} \\
&+ \frac{1}{2}\sum_{n\neq n'}\frac{Z_nZ_{n'}}{|\boldsymbol{R}_n-\boldsymbol{R}_{n'}|},
\end{aligned}
\tag{2.26}
$$

と表わされる．ここで，

$$
\begin{aligned}
V_{ps}^{\boldsymbol{G}\boldsymbol{G}'} &= \sum_n e^{i(\boldsymbol{G}'-\boldsymbol{G})\cdot\boldsymbol{R}_n}\sum_l V_{ps,l}^{\boldsymbol{G}\boldsymbol{G}'} \\
V_{ps,l}^{\boldsymbol{G}\boldsymbol{G}'} &= \int d\boldsymbol{r}e^{-i(\boldsymbol{k}+\boldsymbol{G})\cdot\boldsymbol{r}}V_{ps,l}(\boldsymbol{r})\hat{P}_l(\boldsymbol{r})e^{i(\boldsymbol{k}+\boldsymbol{G}')\cdot\boldsymbol{r}},
\end{aligned}
\tag{2.27}
$$

である．式 (2.26) を $(C_i^{\boldsymbol{G}})^*$について変分すると，Kohn-Sham 方程式の逆格子空間での表式が得られて，

$$
\begin{aligned}
\sum_{\boldsymbol{G}'}H^{\boldsymbol{G}\boldsymbol{G}'}C_i^{\boldsymbol{G}'} &= \epsilon_iC_i^{\boldsymbol{G}}, \\
H^{\boldsymbol{G}\boldsymbol{G}'} &= \frac{1}{2}|\boldsymbol{k}+\boldsymbol{G}|^2\delta_{\boldsymbol{G}\boldsymbol{G}'}+V_{Coul}(\boldsymbol{G}-\boldsymbol{G}')+\epsilon_{xc}(\boldsymbol{G}-\boldsymbol{G}')+V_{ps}^{\boldsymbol{G}\boldsymbol{G}'},
\end{aligned}
\tag{2.28}
$$

となる．一方，式 (2.26) を$\boldsymbol{R}_n$について変分すると n 番めの原子に働く力$\boldsymbol{f}_n$が，

$$
\begin{aligned}
\boldsymbol{f}_n &= -\frac{\partial}{\partial\boldsymbol{R}_n}E_{LDF} \\
&= -\sum_{n\neq n'}\frac{Z_nZ_{n'}(\boldsymbol{R}_n-\boldsymbol{R}_{n'})}{|\boldsymbol{R}_n-\boldsymbol{R}_{n'}|^3} \\
&- \sum_i\sum_{\boldsymbol{G}\boldsymbol{G}'}i(\boldsymbol{G}'-\boldsymbol{G})\sum_l V_{ps,l}^{\boldsymbol{G}\boldsymbol{G}'}(C_i^{\boldsymbol{G}})^*C_i^{\boldsymbol{G}'}e^{i(\boldsymbol{G}'-\boldsymbol{G})\cdot\boldsymbol{R}_n}
\end{aligned}
\tag{2.29}
$$

と求まる．これらの式 (2.28)-(2.29) を用いると，波動関数と原子の運動方程式 (2.7)-(2.8) の逆格子空間での表式は，

$$
\mu\ddot{C}_i^{\boldsymbol{G}}(t) = -\sum_{\boldsymbol{G}'}H^{\boldsymbol{G}\boldsymbol{G}'}(t)C_i^{\boldsymbol{G}'}+\sum_j\Lambda_{ij}C_j^{\boldsymbol{G}}(t) \tag{2.30}
$$

$$
\ddot{\boldsymbol{R}}_n(t) = \boldsymbol{f}_n \tag{2.31}
$$

と表わされる．この運動方程式の時間発展は Verlet のアルゴリズム [9] により，

$$
\begin{aligned}
C_i^{\boldsymbol{G}}(t+\Delta t) &= -C_i^{\boldsymbol{G}}(t-\Delta t)+2C_i^{\boldsymbol{G}}(t) \\
&+ \frac{(\Delta t)^2}{\mu}\left[\sum_{\boldsymbol{G}'}-H^{\boldsymbol{G}\boldsymbol{G}'}(t)C_i^{\boldsymbol{G}'}(t)+\sum_j\Lambda_{ij}C_j^{\boldsymbol{G}}(t)\right]
\end{aligned}
\tag{2.32}
$$

$$
\boldsymbol{R}_n(t+\Delta t) = -\boldsymbol{R}_n(t-\Delta t)+2\boldsymbol{R}_n(t)-\frac{(\Delta t)^2}{M_n}\boldsymbol{f}_n \tag{2.33}
$$

という差分方程式を数値的に解いて求められる．

3. プログラムの概要

3.1 主プログラム

Fig. 3.1に主プログラムの流れを示す．左側に，副プログラムの名前を呼ばれる順番に示した．右側はそれぞれの副プログラムの役割である．副プログラム reclat は与えられた原子構造から逆格子ベクトルを計算する．副プログラム cam は初めての計算のときに平面波の初期値等を与える．副プログラム ewald は，イオンどうしの静電エネルギーを計算するのに必要なエバルド和を計算する．これらの中で重要な副プログラムは samel，setfor，ugokas の 3 つである．

reclat	逆格子ベクトル$\boldsymbol{G}$の作成
↓	
cam	初期値の設定
↓	
ewald	エバルド和の計算
↓	
$\boxed{\text{samel}}$	電子系の CP
↓	
$\boxed{\text{setfor}}$	原子に働く力の計算
↓	
wrgeom	原子配置の書き込み
↓	
fbchk	力のバランスのチェック
↓	
chgdns	電荷密度の計算
↓	
savedd	計算結果の保存
↓	
$\boxed{\text{ugokas}}$	原子系及び電子系の CP

Fig. 3.1 メインプログラムの流れ

カー・パリネロ法本来のスキームでは，電子系と原子系を同時に式 (2.30)-(2.31) に従って時間発展させていくのだが，その様にすると Born-Oppenheimer (BO) 面から電子系がずれて行き収束しなくなることがある．そこでこのプログラムでは電子系が BO 面からずれてきた時に，電子系だけにカー・パリネロ法を適用してクエンチさせている．副プログラム samel がこれをおこなっている．副プログラム setfor は文字どおり，各原子に働く力を計算し設定する．副プログラム ugokas は，時間発展方程式 (2.30)-(2.31) に従って電子系と原子系を動かして分子動力学を行う

部分である．

3.2 副プログラム samel

Fig. 3.2に副プログラム samel の流れを示す．

副プログラム fermi はフェルミ分布に従って各軌道の占有電子数をきめる．副プログラム matset2 は，引数が 0 の時はベッセル関数を含む積分式 (2.25) の計算を行い，結果は変数 $bessel(l, ig, ig', it)$ に保存される．この値は擬ポテンシャルの逆格子空間での表示 $V_{ps}^{\boldsymbol{GG'}}$ を求める際に使用される．副プログラム matset2 の引数が 1 の時は，式 (2.30) に現れる $\sum_{\boldsymbol{G'}} H^{\boldsymbol{GG'}} C_i^{\boldsymbol{G'}}$ の擬ポテンシャルによる部分 $\sum_{\boldsymbol{G'}} V_{ps}^{\boldsymbol{GG'}} C_i^{\boldsymbol{GG'}}$ の計算を行う．結果の実部と虚部はそれぞれ，$hcgr(ig, ib)$ および $hcgi(ig, ib)$ に保存される．副プログラム matset1 では $\sum_{\boldsymbol{G'}} H^{\boldsymbol{GG'}} C_i^{\boldsymbol{G'}}$ のクーロン相互作用と交換相関相互作用による部分を計算し，結果の実部と虚部をそれぞれ，変数 $hvxr(ig, ib)$ と $hvxi(ig, ib)$ に保存する．副プログラム elenergy では全系のエネルギーを計算する．

次の副プログラム nxtwfq が電子系のみの CP 法を行い，電子の波動関数を BO 面にクエンチさせるプログラムである．ここでは先ず副プログラム mkhc がよばれる．副プログラム mkhc は $\sum_{\boldsymbol{G'}} H^{\boldsymbol{GG'}} C_i^{\boldsymbol{G'}}$ を計算する部分で，そのために副プログラム fermi から matset1 を呼びだし，計算結果の実部と虚部をそれぞれ $rhcr(ig, ib)$ と $rhci(ig, ib)$ に保存する．制御が副プログラム nxtwfq に戻ると，$\sum_{\boldsymbol{G'}} H^{\boldsymbol{GG'}} C_i^{\boldsymbol{G'}}$ を元に新しい波動関数を次式に従って求める．

$$\begin{aligned} C_i\ \boldsymbol{G}(t+\delta t) &= C_i^{\boldsymbol{G}}(t) + \alpha(C_i^{\boldsymbol{G}}(t) - C_i^{\boldsymbol{G}}(t-\delta t)) \\ &+ \frac{(\delta t)^2}{\mu}\{-\sum_{\boldsymbol{G'}} H^{\boldsymbol{GG'}} C_i^{\boldsymbol{G'}}(t) + \Lambda_{ii} C_i^{\boldsymbol{G}}(t)\}. \end{aligned} \tag{3.1}$$

この式と Verlet アルゴリズムによる式 (2.30) を比較すると，ここではαを収束を早める加速係数として扱っており，またΛは対角項のみを用いている．従ってこうして得られた新しい波動関数の規格直交条件は満たされておらず，そのため次にシュミット法により規格直交化する．

この副プログラム nxtwfq は電子系の波動関数が十分クエンチされ BO 面に近づくまでくり返し実行される．波動関数がどの程度クエンチされたかは，

$$\Delta_i = \sum_{\boldsymbol{G}} \left| \Lambda_{ii} C_i^{\boldsymbol{G}} - \sum_{\boldsymbol{G'}} H^{\boldsymbol{GG'}} C_i^{\boldsymbol{G'}} \right|^2 \tag{3.2}$$

で定義される残差がどの程度小さくなったかで評価する．

3.3 副プログラム setfor

Fig. 3.3に副プログラム setfor の流れを示す．この副プログラムは各原子に働く力を計算する．各原子に働く力は，式 (2.29) に示したようにイオン－イオンの静電エネルギーによる部分と擬ポテンシャルによる部分からなる．副プログラム ewald は静電エネルギーによる部分をエバルドの方法を用いて計算し，結果を配列 fcc に保存する．擬ポテンシャルによる寄与は副プログラム

sameli

fermi

matset2(0)

$$bessel(l,ig,ig',it) \leftarrow \int_0^\infty dr \sum_{i=1}^{3} (A_i^l + r^2 A_{i+3}^l) e^{-\alpha_i^l r^2} j_l(|\mathbf{k}+\mathbf{G}_i|r) j_l(|\mathbf{k}+\mathbf{G}_i|r)$$

matset2(1)

$$hcgr(ig,ib) \leftarrow \mathrm{Re}(\sum_{\mathbf{G}'} V_{ps}^{\mathbf{GG}'} C_i^{\mathbf{G}'})$$

$$hcgi(ig,ib) \leftarrow \mathrm{Im}(\sum_{\mathbf{G}'} V_{ps}^{\mathbf{GG}'} C_i^{\mathbf{G}'})$$

matset1

$$hvxr(ig,ib) \leftarrow \mathrm{Re}(\sum_{\mathbf{G}'} [V_{coul}(\mathbf{G}-\mathbf{G}') + \varepsilon_{xc}(\mathbf{G}-\mathbf{G}')] C_i^{\mathbf{G}'}$$

$$hvxi(ig,ib) \leftarrow \mathrm{Im}(\sum_{\mathbf{G}'} [V_{coul}(\mathbf{G}-\mathbf{G}') + \varepsilon_{xc}(\mathbf{G}-\mathbf{G}')] C_i^{\mathbf{G}'}$$

elenergy

Calculation of total energy

Enter into CP iteration for wavefunctions

(a)

nxtwfq — mkhc — fermi

matset2(0)

$bessel(l,ig,ig',it)$

$$\leftarrow \int_0^\infty dr \sum_{i=1}^{3} (A_i^l + r^2 A_{i+3}^l) e^{-\alpha_i^l r^2} j_l(|\mathbf{k}+\mathbf{G}_i|r) j_l(|\mathbf{k}+\mathbf{G}_i|r)$$

matset2(1)

$$hcgr(ig,ib) \leftarrow \mathrm{Re}(\sum_{\mathbf{G}} V_{ps}^{\mathbf{GG'}} C_i^{\mathbf{G}})$$

$$hcgi(ig,ib) \leftarrow \mathrm{Im}(\sum_{\mathbf{G}} V_{ps}^{\mathbf{GG'}} C_i^{\mathbf{G}})$$

matset1

$$hvxr(ig,ib) \leftarrow \mathrm{Re}(\sum_{\mathbf{G}} [V_{coul}(\mathbf{G}-\mathbf{G'}) + \varepsilon_{xc}(\mathbf{G}-\mathbf{G'})] C_i^{\mathbf{G'}}$$

$$hvxi(ig,ib) \leftarrow \mathrm{Im}(\sum_{\mathbf{G}} [V_{coul}(\mathbf{G}-\mathbf{G'}) + \varepsilon_{xc}(\mathbf{G}-\mathbf{G'})] C_i^{\mathbf{G'}}$$

$$rhcr(ig,ib) \leftarrow \mathrm{Re}\left(\sum_{\mathbf{G'}} H^{\mathbf{GG'}} C_i^{\mathbf{G'}}\right) = \mathrm{Re}\left(\sum_{\mathbf{G'}} \left[\frac{1}{2}|\mathbf{G}|^2 + V_{coul}(\mathbf{G}-\mathbf{G'}) + \varepsilon_{xc}(\mathbf{G}-\mathbf{G'})\right] C_i^{\mathbf{G'}}\right)$$

$$rhci(ig,ib) \leftarrow \mathrm{Im}\left(\sum_{\mathbf{G'}} H^{\mathbf{GG'}} C_i^{\mathbf{G'}}\right) = \mathrm{Im}\left(\sum_{\mathbf{G'}} \left[\frac{1}{2}|\mathbf{G}|^2 + V_{coul}(\mathbf{G}-\mathbf{G'}) + \varepsilon_{xc}(\mathbf{G}-\mathbf{G'})\right] C_i^{\mathbf{G'}}\right)$$

$$C_i^{\mathbf{G}}(t+\Delta t) = C_i^{\mathbf{G}}(t) + \alpha(C_i^{\mathbf{G}}(t) - C_i^{\mathbf{G}}(t-\Delta t)) + \frac{(\Delta t)^2}{\mu}(-\sum_{\mathbf{G'}} H^{\mathbf{GG'}} C_i^{\mathbf{G'}}(t) + \Lambda_{ii} C_i^{\mathbf{G}}(t))$$

orthogonalization of $C_i^{\mathbf{G}}(t+\Delta t)$ by Schmidt method

$$cgjr(ig,ib) = \mathrm{Re}(C_i^{\mathbf{G}}(t+\Delta t))$$

$$cgji(ig,ib) = \mathrm{Im}(C_i^{\mathbf{G}}(t+\Delta t))$$

$$c1gr(ig,ib) = \mathrm{Re}(C_i^{\mathbf{G}}(t))$$

$$c1gi(ig,ib) = \mathrm{Im}(C_i^{\mathbf{G}}(t))$$

samelq

(b)

Fig. 3.2 副プログラム samel の流れ

matset2(0)

$$bessel(l,ig,ig',it) \leftarrow \int_0^\infty dr \sum_{i=1}^{3} (A_i^l + r^2 A_{i+3}^l) e^{-\alpha_i^l r^2} j_l(|\mathbf{k}+\mathbf{G}_i|r) j_l(|\mathbf{k}+\mathbf{G}_i|r)$$

ewald

$$\left.\begin{matrix} fcc(ia,it,1) \\ fcc(ia,it,2) \\ fcc(ia,it,3) \end{matrix}\right\} \leftarrow -\frac{\partial}{\partial \mathbf{R}_n} \sum_{n \neq n'} \frac{Z_n Z_{n'}}{|\mathbf{R}_n - \mathbf{R}_{n'}|^3}$$

fermi

Calculate occupation numer according to Fermi distribution function

matset2(2)

$$\left.\begin{matrix} fec(ia,it,1) \\ fec(ia,it,2) \\ fec(ia,it,3) \end{matrix}\right\} \leftarrow -\frac{\partial}{\partial \mathbf{R}_n} E_{ps} = -\sum_{\mathbf{GG'}} i(\mathbf{G'}-\mathbf{G}) V_{ps}^{\mathbf{GG'}} (C_i^{\mathbf{G}})^* C_i^{\mathbf{G'}} e^{i(\mathbf{G}-\mathbf{G'})\mathbf{R}_n}$$

matset1

$$hvxr(ig,ib) \leftarrow \mathrm{Re}(\sum_{\mathbf{G'}} [V_{coul}(\mathbf{G}-\mathbf{G'}) + \varepsilon_{xc}(\mathbf{G}-\mathbf{G'})] C_i^{\mathbf{G'}}$$

$$hvxi(ig,ib) \leftarrow \mathrm{Im}(\sum_{\mathbf{G'}} [V_{coul}(\mathbf{G}-\mathbf{G'}) + \varepsilon_{xc}(\mathbf{G}-\mathbf{G'})] C_i^{\mathbf{G'}}$$

$$fox(ia,it) = fcc(ia,it,1) + fec(ia,it,1)$$

$$foy(ia,it) = fcc(ia,it,2) + fec(ia,it,2)$$

$$foz(ia,it) = fcc(ia,it,3) + fec(ia,it,3)$$

Fig. 3.3 副プログラム setfor の流れ

matset2(2) により計算されその結果は配列 fec に保存される．最後に両者による寄与はたし合わされて，各原子に働く力は配列 fox，foy，fozに保存される．

3.4 副プログラム ugokas

Fig. 3.4に副プログラム ugokas の流れを示す．図中の副プログラム setfor，wrgeom，elenergy，onchk については既に説明したのでここでは省略する．この副プログラムで重要な部分は副プログラム verlet と nextwf である．副プログラム verlet は，副プログラム setfor で計算された各原子に働く力をもとに Verlet アルゴリズムを用いて実際に原子位置を時間発展させていく部分である．副プログラム nextwfq は Verlet アルゴリズムを用いて波動関数を時間発展させていく部分である．副プログラム samel のなかで使用される nxtwfq とは異なることに気をつけていただきたい．副プログラム nextwf のなかでは，配列$\sum_{\boldsymbol{G}'} H^{\boldsymbol{G}\boldsymbol{G}'} C_i^{\boldsymbol{G}'}$ を計算するために先ず副プログラム mkhc が呼ばれる．副プログラム mkhc については既に説明したのでここでは省略する．次にラグランジュの未定係数Λ_{ij}を計算するために副プログラム abxstp が呼ばれる．求まった未定係数は配列 $x0r$，$x0i$ に保存される．

制御が副プログラム nextwf に戻ると，Verlet のアルゴリズムに従って新しい波動関数が計算される．新しい波動関数は規格直交性を拘束条件としたラグランジュアンをもとに計算されているため，改めてシュミット法などの直交化を行う必要はない．波動関数系の BO 面からのずれは副プログラム nxtwfq と同様に，

$$\Delta_i = \sum_{\boldsymbol{G}} \left| \sum_j \Lambda_{ij} C_j^{\boldsymbol{G}} - \sum_{\boldsymbol{G}'} H^{\boldsymbol{G}\boldsymbol{G}'} C_i^{\boldsymbol{G}'} \right|^2 \tag{3.3}$$

で定義される残差により評価される．もし残差が一定値より大きいときは，副プログラム samel が呼ばれ，波動関数を BO 面にクエンチする．

一連の波動関数と原子位置のカー・パリネロ法に基づいた時間発展は，指定した回数だけ行われる．以上がプログラムの流れである．

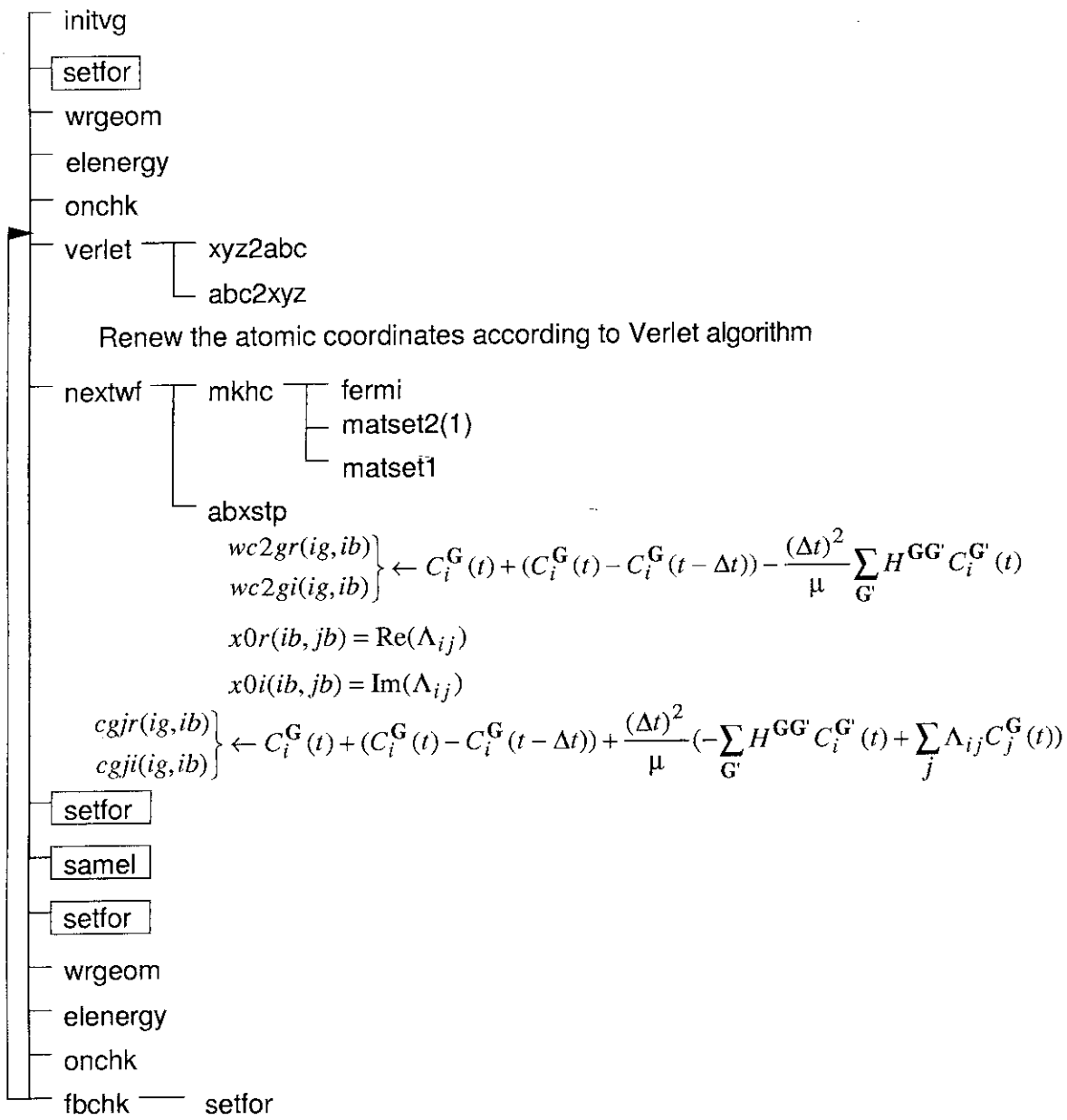

Fig. 3.4 副プログラム ugokas の流れ

4. ベクトル並列化

4.1 プログラムの解析

並列化によりプログラムを高速化するためには，まず負荷の高い部分を探し，その部分から並列化していくのが効率的である．各副プログラム毎の処理時間の全体に占める割合を Table 4.1に示す．これは共有メモリー型ベクトル並列スーパーコンピュータ Monte4 上のアナライザーを用いて計測したもので，VPP での実態を示しているわけではないが，同じベクトル並列機ということで大体の様子は把握できると思われる．VPP 上にもサンプラーと呼ばれる負荷分布を計測するツールが用意されている．これは一定時間毎に CPU がプログラムのどの部分を処理しているかをサンプリングすることができるが，このツールは計測のためのオーバーヘッドが大きすぎて実際の負荷分散をあまり反映していない事が経験上解ったため，敢えて Monte-4 上のアナライザーの結果を示した．測定に用いた系は，単位格子に 8 個の Si 原子を含む結晶構造から各原子をすこしずらした系で，Camp-Atami にサンプルとして付属する入力データの一つである．カットオフエネルギーは 3Ry とし，原子を動かすステップを 10 回実行した．Table 4.1から，副プログラム

Table 4.1 各副プログラムの負荷の全体に示す割合

Name	CPU time (sec)	Ratio (%)	V. op ratio (%)
fft3d	87.861	78.1	99.43
calcfec	6.313	5.6	98.31
matset1	5.594	5.0	91.50
calchc	4.945	4.4	97.96
mkvps	2.099	1.9	96.68
chgdns	1.857	1.7	90.47
avxc	1.090	1.0	98.64
evald	0.807	0.7	63.63
erflocal	0.668	0.6	0.00
reclat	0.241	0.2	0.04
xfft3e	0.231	0.2	7.02
mkbessel	0.124	0.1	34.29
onchk	0.092	0.1	73.23
nextwf	0.082	0.1	87.76
schmidt	0.061	0.1	97.03
abxstp	0.057	0.1	97.41

fft3d だけで全 CPU 時間の 8 割近くを消費していることが解る．この副プログラムは計算の中の色々な場所で行われる 3 次元の高速フーリエ変換を受け持つプログラムである．従って，この副

プログラムを並列化すれば良いように思われる．しかしながら，高速フーリエ変換ではフーリエ変換を行う軸に対しては全てのデータを持っている必要があり，分散メモリー型の並列計算機では必ず全ての CPU 間でのデータ転送が発生し並列化による計算時間の短縮には限度がある．一方，並列化の観点からはできるだけ粒度を大きくして並列化したほうが一般に効率が良い．そこで fft3d を含んだより外側のルーティンでの並列化をできるだけ試みた．

Table 4.1を見ると，fft3d，calcfec，matset1，calchc，mkvps，chgdns6 つの副プログラムで全 CPU の負荷の 97%を占めていることが解る．これらは，全て副プログラム matset1 か matset2 から呼ばれるプログラムである．従って matset1 と matset2 の 2 つを並列化できれば良いことが解る．そこでこの 2 つの副プログラムを中心に並列化を行った．以下の節ではどのようにこの 2 つの副プログラムが並列化できたか見ていくことにする．

プログラムの並列化では，できるだけ高速フーリエ変換の外側のルーティンで並列化するよう試みたが，高速化のためには高速フーリエ変換自体を並列化しなければいけない部分もある．そこで高速フーリエ変換の並列化も行い，プログラム中では並列版と非並列版のフーリエ変換を場所に応じて使い分けることにした．第 4.4 節に高速フーリエ変換の並列化についても述べる．

4.2 副プログラム matset2 の並列化

前述のように副プログラム matset2 は，式 (2.28) の$\sum_{\boldsymbol{G}'} H^{\boldsymbol{G}\boldsymbol{G}'} C_i^{\boldsymbol{G}'}$において擬ポテンシャルによる寄与の部分を計算するプログラムである．Fig. 4.1に副プログラム matset2 の概略を示す．この副プログラムは引数 *icalc* を持っていて，その値により異なる処理をして親ルーティンに結果を返す．

引数 *icalc* が 0 のときは，副プログラム mkbessel を呼び出し，擬ポテンシャルの計算に必要な積分 (2.25) を計算して変数 *bessel*に保存する．

引数 *icalc* が 1 のときは，副プログラム calchc を呼び出し，式 (2.28) の$\sum_{\boldsymbol{G}'} H^{\boldsymbol{G}\boldsymbol{G}'} C_i^{\boldsymbol{G}'}$において擬ポテンシャルによる寄与の部分を計算する．副プログラム calchc は先ず副プログラム mkvps を呼び，擬ポテンシャル $V_{ps,l}^{\boldsymbol{G}\boldsymbol{G}'}$を計算する．この時，副プログラム mkbessel で計算された変数 *bessel* の値を使用する．次に calchc で計算した擬ポテンシャルをもとに$\sum_{\boldsymbol{G}'} H^{\boldsymbol{G}\boldsymbol{G}'} C_i^{\boldsymbol{G}'}$の擬ポテンシャルによる寄与の部分を計算し，その実数部と虚数部をそれぞれ変数 *hcgr*，と *hcgi* に保存する．

引数 *icalc* が 2 のときは，副プログラム calcfec を呼び出し，式 (2.28) の$\sum_{\boldsymbol{G}'} H^{\boldsymbol{G}\boldsymbol{G}'} C_i^{\boldsymbol{G}'}$において擬ポテンシャルによる寄与の部分を計算し，同時に各原子に働く力のうち，擬ポテンシャルによる部分を計算する．副プログラム calcfec は，上に述べた calchc とほとんど同じである．異なる点は calcfec が，擬ポテンシャルエネルギーの座標 $\{\boldsymbol{R}_n\}$ についての偏微分を計算し n 番目の原子に働く力のうち擬ポテンシャルによる寄与を求めることである．

以下では，副プログラム matset2 を並列化することを考える．副プログラム matset2 は，基本的には引数 *icalc* に従って，副プログラム mkbessell，calchc，calcfec を呼んでいるだけである．副プログラム mkbessel を除いて，calchc，calcfec は逆格子ベクトル$\boldsymbol{G}$についての DO ループの中で呼ばれている．この$\boldsymbol{G}$についてのループは，それぞれ独立に実行することができるのでこの部分

subroutine matset2(icalc)

if (icalc.eq.0) then

— mkbessel

$$bessel(l,ig,ig',it) \leftarrow \int_0^\infty dr \sum_{i=1}^{3} (A_i^l + r^2 A_{i+3}^l) e^{-\alpha_i^l r^2} j_l(|\mathbf{k}+\mathbf{G}_i|r) j_l(|\mathbf{k}+\mathbf{G}_i|r)$$

if (icalc.eq.1) then

— calchc — mkvps

$$vps(jg) \leftarrow \sum_l V_{ps,l}^{\mathbf{GG'}}$$

$$psr(ig) \leftarrow \mathrm{Re}(V_{ps}^{\mathbf{GG'}}) = \mathrm{Re}(\sum_n e^{i(\mathbf{G'}-\mathbf{G})\mathbf{R}_n} \sum_l V_{ps,l}^{\mathbf{GG'}})$$

$$psi(ig) \leftarrow \mathrm{Im}(V_{ps}^{\mathbf{GG'}}) = \mathrm{Im}(\sum_n e^{i(\mathbf{G'}-\mathbf{G})\mathbf{R}_n} \sum_l V_{ps,l}^{\mathbf{GG'}})$$

$$hcgr(ig,ib) \leftarrow \mathrm{Re}(\sum_{\mathbf{G'}} V_{ps}^{\mathbf{GG'}} C_i^{\mathbf{G'}})$$

$$hcgi(ig,ib) \leftarrow \mathrm{Im}(\sum_{\mathbf{G'}} V_{ps}^{\mathbf{GG'}} C_i^{\mathbf{G'}})$$

if (icalc.eq.2) then

— calcfec — mkvps

$$vps(jg) \leftarrow \sum_l V_{ps,l}^{\mathbf{GG'}}$$

$$psr(ig) \leftarrow \mathrm{Re}(V_{ps}^{\mathbf{GG'}}) = \mathrm{Re}(\sum_n e^{i(\mathbf{G'}-\mathbf{G})\mathbf{R}_n} \sum_l V_{ps,l}^{\mathbf{GG'}})$$

$$psi(ig) \leftarrow \mathrm{Im}(V_{ps}^{\mathbf{GG'}}) = \mathrm{Im}(\sum_n e^{i(\mathbf{G'}-\mathbf{G})\mathbf{R}_n} \sum_l V_{ps,l}^{\mathbf{GG'}})$$

$$hcgr(ig,ib) \leftarrow \mathrm{Re}(\sum_{\mathbf{G'}} V_{ps}^{\mathbf{GG'}} C_i^{\mathbf{G'}})$$

$$hcgi(ig,ib) \leftarrow \mathrm{Im}(\sum_{\mathbf{G'}} V_{ps}^{\mathbf{GG'}} C_i^{\mathbf{G'}})$$

$$\left.\begin{matrix} fec(ia,it,1) \\ fec(ia,it,2) \\ fec(ia,it,3) \end{matrix}\right\} \leftarrow -\frac{\partial}{\partial \mathbf{R}_n} E_{ps} = -\sum_{\mathbf{GG'}} i(\mathbf{G'}-\mathbf{G}) V_{ps}^{\mathbf{GG'}} C_i^{\mathbf{G}*} C_i^{\mathbf{G'}} e^{i(\mathbf{G}-\mathbf{G'})\mathbf{R}_n}$$

Fig. 4.1 副プログラム matset2 の流れ

を並列化することができる．

副プログラム calchc や calcfec による計算結果は変数 *hcgr*, *hcgi*, *fec* に保存されるので，並列に実行した場合はこれらの変数に対して各ノードで統合を行う必要がある．以下に上記の方針に従って，並列化を行ったプログラムソースの概略を示す．

```
      subroutine matset2(icalc)
      parameter (npe=4)
!xocl processor pp(npe)
!xocl subprocessor pp2(npe)=pp(1:npe)
!xocl index partition bp=(pp2, index=1:lband, part=band)
!xocl index partition gp=(pp2, index=1:lpl, part=band)
      do it=1,ntype
        do ia=1,natom(it)
          fec(ia,it,1)=0
          fec(ia,it,2)=0
          fec(ia,it,3)=0
        enddo
      enddo
c:
      do ib=1,nband
        do ig=1,npw
          hcgr(ig,ib)=0
          hcgi(ig,ib)=0
        enddo
      enddo
c:
      if (icalc.eq.0) goto 1000
      if (icalc.eq.1) goto 2000
      if (icalc.eq.2) goto 3000
c:
 1000 continue
      call mkbessel
      return
c:
 2000 continue          parallel process for ig
!xocl spread do / gp  <----------------------------+
      do ig=1,nplw                                  |
        call calchc(nplw,ig)                        |
      enddo                                         |
!xocl end spread sum (hcgr), sum (hcgi)  <----------+
      return
c:
 3000 continue          parallel precess for ig
!xocl spread do / gp  <----------------------------+
      do ig=1,nplw                                  |
        call calcfec(nplw,ig)                       |
      enddo                                         |
!xocl end spread sum (fec), sum(hcgr), sum(hcgi)  <-+
      return
```

4.3 副プログラム matset1 の並列化

副プログラム matset1 は式 (2.28) の$\sum_{\boldsymbol{G}'} H^{\boldsymbol{G}\boldsymbol{G}'} C_i^{\boldsymbol{G}'}$において，クーロン相互作用と交換相関相互作用による部分を計算するプログラムである．Fig. 4.2にプログラムの流れを示す．副プログラム matset1 は，先ず副プログラム chgdns 呼ぶ．副プログラム chgdns は，波動関数の平面波展開係数$C_i^{\boldsymbol{G}}$からフーリエ変換により実空間での波動関数$\psi_i(\boldsymbol{r})$を求め，式 (2.4) により実空間での電子密度$\rho(\boldsymbol{r})$を計算し，結果を配列 *rho* に保存して親ルーティンに戻す．

次に matset1 は副プログラム svxc を呼び出して，実空間での交換相関ポテンシャル$\epsilon_{xc}(\boldsymbol{r})$を

subroutine matset1

— chgdns — fft3d $\Psi_{i(r)} = \frac{1}{\sqrt{\Omega}} \sum_{\mathbf{G}} e^{i\mathbf{G}\cdot\mathbf{r}} C_i^{\mathbf{G}}$

$$rho(i) \leftarrow \rho(\mathbf{r}) = \sum_i n_i |\Psi_{i(r)}|^2$$

— svxc

$$vxc(i) \leftarrow \varepsilon_{xc}(\mathbf{r})$$

— fft3d

$$vhxcr(i) \leftarrow \mathrm{Re}(\varepsilon_{xc}(\mathbf{G})) = \mathrm{Re}[\int d\mathbf{r} e^{-i\mathbf{G}\cdot\mathbf{r}} \varepsilon_{xc}(\mathbf{r})]$$

$$vhxci(i) \leftarrow \mathrm{Im}(\varepsilon_{xc}(\mathbf{G})) = \mathrm{Im}[\int d\mathbf{r} e^{-i\mathbf{G}\cdot\mathbf{r}} \varepsilon_{xc}(\mathbf{r})]$$

— fft3d

$$\rho(\mathbf{G}) = \int d\mathbf{r} e^{-i\mathbf{G}\cdot\mathbf{r}} \rho(\mathbf{r})$$

$$V_{Coul}(\mathbf{G}) = \frac{4\pi\rho(\mathbf{G})}{|\mathbf{G}|^2}$$

$$vhxcr(i) \leftarrow \mathrm{Re}(\varepsilon_{xc}(\mathbf{G})) = \mathrm{Re}[\int d\mathbf{r} e^{-i\mathbf{G}\cdot\mathbf{r}} \varepsilon_{xc}(\mathbf{r})]$$

$$vhxci(i) \leftarrow \mathrm{Im}(\varepsilon_{xc}(\mathbf{G})) = \mathrm{Im}[\int d\mathbf{r} e^{-i\mathbf{G}\cdot\mathbf{r}} \varepsilon_{xc}(\mathbf{r})]$$

— fft3d

$$vhr(i) \leftarrow \varepsilon_{xc}(\mathbf{r}) = \sum_{\mathbf{G}} e^{i\mathbf{G}\cdot\mathbf{r}} \varepsilon_{xc}(\mathbf{G})$$

$$hvxr(ig,ib) \leftarrow \mathrm{Re}(\sum_{\mathbf{G}'} [V_{coul}(\mathbf{G}-\mathbf{G}') + \varepsilon_{xc}(\mathbf{G}-\mathbf{G}')] C_i^{\mathbf{G}'}$$

$$hvxi(ig,ib) \leftarrow \mathrm{Im}(\sum_{\mathbf{G}'} [V_{coul}(\mathbf{G}-\mathbf{G}') + \varepsilon_{xc}(\mathbf{G}-\mathbf{G}')] C_i^{\mathbf{G}'}$$

Fig. 4.2 副プログラム matset1 の流れ

計算し変数 vxc に保存する．この実空間での交換相関ポテンシャルは，副プログラム fft3d によりフーリエ変換され逆格子空間での関数$\epsilon_{xc}(\boldsymbol{G})$ となり，その実部と虚部はそれぞれ配列 $vhxcr$, $vhxci$ に保存される．

さらに matset1 は上で求めた実空間の電子密度$\rho(\boldsymbol{r})$ をフーリエ変換して逆格子空間での電子密度$\rho(\boldsymbol{G})$ を計算する．逆格子空間でのクーロンポテンシャルは，この電子密度を用いて式 (2.20) で与えられる．ここで得られた逆格子空間でのクーロンポテンシャルは，上で得られた相関交換ポテンシャルとたし合わされてその実部と虚部はそれぞれ変数 $vhxcr$と $vhxci$ に保存される．逆格子空間でのクーロンポテンシャルはフーリエ変換により実空間に移され，変数 vhrに保存される．これは他の副プログラムで系全体のエネルギーを求めるときに使用される．

最後に副プログラム matset1 は，$\sum_{\boldsymbol{G}'} H^{\boldsymbol{G}\boldsymbol{G}'} C_i^{\boldsymbol{G}'}$ のクーロン相互作用と交換相関相互作用による部分を計算し，その実部と虚部をそれぞれ変数 $hvxr$と $hvxi$ に保存する．

以下では，副プログラム matset1 の並列化について述べる．副プログラム matset1 での処理のほとんどの部分は，フーリエ変換に費やされている．フーリエ変換の副プログラム fft3d は 4 箇所で呼ばれているが，特に副プログラム chgdns から呼ばれている fft3d はバンドインデックス i によるループの中で呼ばれていて，この部分は並列に実行できる．そこでこの高速フーリエ変換を並列に実行し，最後に電子密度 rho について各ノードで統合してやればよい．

一方，副プログラム matset1 から直接呼ばれる 3 つの fft3d は並列に実行することはできないので，fft3d 自体を並列化する必要がある．そこで並列化してない高速フーリエ変換 fft3d と並列化した高速フーリエ変換 fft3dx の 2 つを用意し，前者は副プログラム chgdns の中で使用し，後者はそれ以外のところで使用するにした．以下に並列化したプログラムソースの概要を示す．高速フーリエ変換の並列化の実際については，次章を参照のこと．

```
      subroutine matset1(nodig, npw, nx,ny, nz)
      parameter (npe=4)
!xocl processor pp(npe)
!xocl subprocessor pp2(npe)=pp(1:npe)
!xocl index partition bp=(pp2, index=1:lband, part=band)
!xocl index partition gp=(pp2, index=1:lpl, part=band)
      do i=1,nmesh
        vhxcr(i) = 0
        vhxci(i) = 0
      enddo
c:
c:    calculate charge density in real space rho(r) -> rho(nmesh)
c:
      call chgdns(npw, kfft1, kfft2, kfft3, rho, .false.)
c:
c:    calculate exchange-correlation Exc(r) -> vxc (nmesh)
c:
      call svxc(rho, nx, ny, nz, vxc, exc)
      do i=1,nmesh
        fftr(i) = vxc(i)
        ffti(i) = 0
      enddo
c:
c:    FFT Exc(r) to Exc(G) -> vhxcr, vhxci
c:!       using parallelized FFT: fft3dx
c:
      call fft3dx(fftr, ffti, nx, ny, nz)
      do ig = 1, npw
        vhxcr(ijk) = fftr(ijk)
```

```
        vhxci(ijk) = ffti(ijk)
      enddo
c:
      do i = 1, nmesh
        fftr(i) = rho(i)
        ffti(i) = 0
      enddo
c:
c:    FFT rho(r) to rho(G) -> fftr, ffti
c:!       using parallelized FFT: fft3dx
c:
      call fft3dx(fftr, ffti, nx, ny, nz, lfftf)
c:
c:    Vcoul(G) = 4*Pi*rho(G)/|G|^2 -> fftr(ijk), ffti(ijk)
c:    Vsoul(G) + Exc(G) -> vhxcr(ijk), vhxci(ijk)
c:
      do ig = 2,npw
        fftr(ijk) = 4*Pi*fftr(ijk)/G^2
        ffti(ijk) = 4*Pi*ffti(ijk)/G^2
        vhxcr(ijk) = vhxcr(ijk)+fftr(ijk)
        vhxci(ijk) = vhxci(ijk)+ffti(ijk)
      endo
c:
c:    FFT Vcoul(G) to Vcoul(r) -> vhr(nmesh)
c:!       using parallelized FFT: fft3dx
c:
      call fft3d(fftr, ffti, nx, ny ,nz, lffti)
c:
c:
c:    Calculate sum_G' H^GG' C_G',
c:
c:
      do ib = 1,nband
        do ig = 1, npw
          hvxr(ig,ib) = 0
          hvxi(ig,ib)
        enddo
      enddo
c:
!xocl spread do / gp
      do ig = 1, npw
        do jg = 1, npw
          do ib = 1, npw
            hvxr(ig,ib) = hvxr(ig,ib)+vhxcr(ijk)*cgjr(jg,ib)
     &                              -vhxci(ijk)*cgji(jg.ib)
            hvxr(ig,ib) = hvxr(ig,ib)+vhxcr(ijk)*cgjr(jg,ib)
     &                              -vhxci(ijk)*cgji(jg.ib)
          enddo
        enddo
      enddo
!xocl end spread sum (hvxr), sum (hvxi)
c:
      return
      end
c:
c:
      subroutine chgdns(npw, nx, ny, nz, arho, ldump)
c:
!xocl processor pp(npe)
!xocl subprocessor pp2(npe)=pp(1:npe)
!xocl index partition bp=(pp2, index=1:lband, part=band)
!xocl index partition gp=(pp2, index=1:lpl, part=band)
c:
      do i=1,nmesh
        arho(i)=0
      enddo
c:
!xocl spread do / bp
      do ib=1,nband
        do i=1,nmesh
          fftr(i)=0
```

```
          ffti(i)=0
        enddo
c:
        do ig=1,npw
          fftr(index)=cgjr(ig,ib)
          ffti(index)=cgji(ig,ib)
        enddo
c:
c:      FFT C(G) to psi(r) -> fftr, ffti
c:
        call fft3d(fftr,ffti,nx,ny,nz,lffti)
c:
        do i=1,nmesh
          arho(i)=arho(i)+( fftr(i)**2+ffti(i)**2 )*2*weg(ib)
        enddo
c:
      enddo
!xocl end spread sum (arho)
      return
      end
```

4.4 高速フーリエ変換の並列化

ここでは，3次元高速フーリエ変換（FFT）のベクトル並列化について述べる．もとのCamp-Atamiに付属のFFTはマニュアルにも書いてあるように非常に遅いので，新たにFFTの副プログラムを作成し差し替えることにした．プログラムはベクトル化したものとベクトル並列化したものと2種類作成した．これは前述のようにFFTの外側でより大きい粒度で並列化できるときは外側で並列化し，それが不可能な部分だけFFTを直接並列化するという指針に立つからである．

VPP用に並列化したFFTは，サブルーティンライブラリーとして提供されているのでこれよりも高速なものを作成することを目的とした．ライブラリーのFFTは，多基底に対応し，またFFTのバタフライ演算に必要な回転因子はそのつど計算する．一方，Camp-Atamiで必要なFFTは，2基底だけであり各軸の要素数も固定である．そこでライブラリーのFFTに比べ，2基底にしぼり回転因子も初めに1度だけ計算するという2つの制限を置くことで高速化をはかった．具体的には，横川が作成した1次元版のisogeometric型のFFT[10]をもとにこれを3次元に拡張し，ベクトル化およびベクトル並列化をおこなった．以下にベクトル並列化の指針を示す．

3次元のフーリエ変換は，3つの軸（x軸，y軸，z軸）についてフーリエ変換を順番に行っていく．各軸に対するフーリエ変換は3次元であることから3重のDOループで計算される．各ループは独立であるが並列化の観点からは，なるべく粒度を大きくするために，一番外側のループを並列化するのがよい．この時，高速フーリエ変換はフーリエ変換する軸については全てのデータを持っていなければならないので，並列化軸をフーリエ変換軸に選ぶことはできない．つまり一番外側のループを並列化し，フーリエ変換はその内側で行うべきである．次にベクトル化の観点からは，一番内側のループをフーリエ変換軸に選ぶべきではない．これは各次元の要素数をnとすると，高速フーリエ変換をおこなう軸のループ長が$log_2 n$で他の軸のnに比べ小さくなると同時にストライドも一定ではなくなるためである．

結果として3次元のFFTのベクトル並列化においては，3重ループのうち1番外側を並列化軸とし，2番めのループで高速フーリエ変換を行い，1番内側のループをベクトル化軸とするのが良いことが解る．

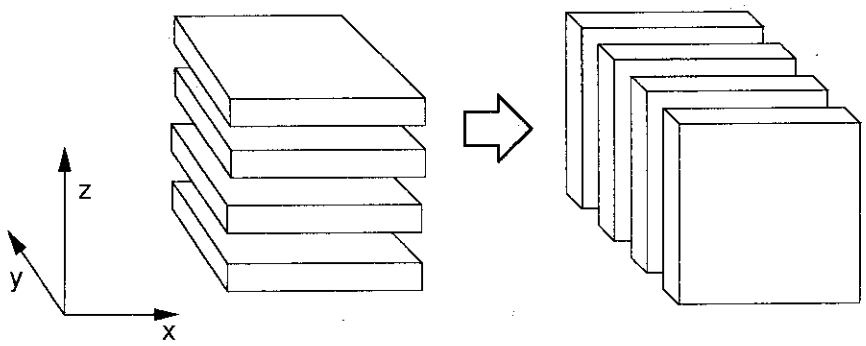

Fig. 4.3 並列版高速フーリエ変換におけるデータ分割の概念図

以下に VPP 用並列版 3 次元 FFT の実際のプログラムソースの概略を示す．配列 u_(ix,iy,iz) がフーリエ変換を行うデータであり結果は u_(ix,iy,iz) に上書きされる．実際は複素数のフーリエ変換であるがここでは簡単のために省略してある．また高速フーリエ変換に伴うバタフライ処理も並列化には関係ないので省略した．

```
      subroutine fft3dx(u,nx,ny,xz)
c:
      parameter(npe=4)
c:
      dimension    u(nfftk1,nfftk2,nfftk3)
      dimension   uz(nfftk1,nfftk2,nfftk3)
      dimension   uy(nfftk1,nfftk2,nfftk3)
      dimension uy_g(nfftk1,nfftk2,nfftk3)
      dimension uz_g(nfftk1,nfftk2,nfftk3)
!xocl processor pp(npe)
!xocl subprocessor pe(npe)=pp(1,npe)
!xocl local  uz  (:,:,/pe), uy  (:,/pe,:)
!xocl gloval uz_g(:,:,/pe), uy_g(:,/pe,:)
      equivalence (uy, uy_g)
      equivalence (uz, uz_g)
c:
!xocl spread do /pe ---------------------------------+
      do iz=1,nz                                     |
        do iy=1,ny --------------------+             | parallel
          do ix=1,nx                   | copy to     | processing
            uz(ix,iy,iz)=u(ix,iy,iz)   | local data  | in z axis
          enddo                        |             |
        enddo -------------------------+             |
c:                                                   |
        do i=1,log2ny --------+                      |
          do ix=1,nx          | butterfly in         |
            uz(ix,*,iz)=...   | y-axis               |
          enddo               |                      |
        enddo ----------------+                      |
c:                                                   |
        do i=1,log2nx --------+                      |
          do iy=1,ny          | butterfly in         |
            uz(*,iy,iz)=...   | x axis               |
```

```
          enddo                 |                        |
        enddo ----------------+                          |
      enddo                                              |
!xocl end spread ------------------------------------+
c:
!xocl spread move /pe -------------------+
      do iz=1,nz                          | transpose
        do iy=1,ny                        | of data
          do ix=1,ny                      |
            uy_g(ix,iy,iz)=uz(ix,iy,iz)   |
          enddo                           |
        enddo                             |
      enddo                               |
!end spread (x)                           |
!movewait(x) ----------------------------+
c:
!xocl spread do /pe ---------------------------------+
      do iy=1,ny                                      |
        do i=1,log2nz --------+                       | parallel
          do ix=1,nx          | butterfly in          | processing
            uy(ix,iy,*)=...   | z axis                | in y axis
          enddo               |                       |
        enddo ----------------+                       |
      enddo                                           |
!xocl end spread ------------------------------------+
!xocl spread move ----------------------+
      do iz=1,nz                        |
        do iy=1,ny                      | unify of the data
          do ix=1,nx                    |
            u(ix,iy,iz)=uy_g(ix,iy,iz)  |
          enddo                         |
        enddo                           |
      enddo                             |
!end spread (x)                         |
!xocl movewait (x) ---------------------+
```

プログラムでは，まず FFT をかける 3 次元データ配列 u(ix,iy,iz) を，z 軸方向に各 PE に分割した分割ローカル変数 u_z(ix,iy,iz) に代入する．そして y 軸に関して高速フーリエ変換を行う．この時 x 軸に関するループは内側にする．同様に x 軸に関する高速フーリエ変換を行う．以上の作業は z 軸に関する DO ループを分割して並列におこなう．次に z 軸方向に高速フーリエ変換を行わなければいけないが，そのためには各 PE で z 軸方向のデータをすべて持っていなければならない．そこで，今まで z 軸方向に分割して各 PE で持っていたデータ u_z(ix,iy,iz) を y 軸方向に分割したデータ u_y(ix,iy,iz) に転置する（ Fig. 4.3参照 ）．転置した後は，z 軸に関する高速フーリエ変換を，y 軸に関する DO ループを分割して各 PE で並列処理する．最後に各 PE で分割して持っているフーリエ変換の終わったデータ u_y(ix,iy,iz) を，重複ローカル変数 u(ix,iy,iz) にコピーして全ての PE で同じデータを持つようにする．

4.5 パラメータ作成プログラム

プログラムを並列化する上では，計算に必要な配列の大きさを正確に把握する必要がある．特にこれは，データ分割を行う時に無駄な計算を行ったりロードバランスを崩さないために重要である．一方，限られたメモリー空間でできるだけ大きな系の計算を行うためにも，配列の大きさ

を必要最小限にとどめることは重要である．そこで配列の大きさを parameter 文で渡すことにして，その parameter 文を入力データから自動的に生成するプログラムを作成した．

使用法は至って簡単で，必要な入力ファイルは実際の計算に使用するものと同じで実行するプログラムが違うだけである．結果は，装置番号 20 番に割り当てられたファイルに書き込まれる．Camp-Atami では，入力ファイルにエネルギーカットオフと FFT で使用するメッシュサイズの両方を記述するが，入力したメッシュサイズがエネルギーカットオフから決められるメッシュサイズに比べ大きい時や小さいときは，メッシュサイズを修正した parameter 文が生成され修正した旨の警告がコメントされる．大きめのメッシュサイズで計算したいときは，parameter 文の該当部分を編集すればよい．ただし小さめには変更できない．並列実行をするときは，PE 数が 1 となっているので実際に使用する PE 数に変更する必要がある．

利用者は，自動生成されたファイルを size.inc と名前を変更して，プログラムをメイクし直せば必要最小限の配列の大きさで最小限のメモリーサイズの実行プログラムが得られる．

5. 性能評価

5.1 全体としての性能

ここでは，Camp-Atami 全体のベクトル並列化による性能向上を評価する．まず，ベクトル最適化による性能向上について見ることにする．ベクトル最適化で行った作業の 1 つは，前述のように FFT ルーティンの書き換えである．これにより 32x32x32 の大きさの 3 次元配列を高速フーリエ変換するときの CPU 時間は約 3.4 倍高速化した．その外のベクトル最適化としては，リストベクトルの導入によりベクトル長を伸ばす，ベクトル化されている DO ループ内でデータアクセスのストライドが 1 になるようにする，ベクトル化できないループをベクトル化できるように書き直す，冗長なコードを削るといった一般的な最適化を随所で行った．これらのベクトル最適化により，CPU 時間で見て約 4.3 倍の高速化がはかれた．計測に用いた系は，単位胞内に 8 個の Si 原子を結晶配置から少しずらした位置に置いたもので，エネルギーカットオフは 3Ry とした．

Table 5.1にベクトル最適化後の各副プログラムの全体に対する負荷分布を示す．ベクトル最適化前の負荷分布を示した Table 4.1 の時と同様に，共有メモリー型ベクトル並列スーパーコンピュータ Monte4 上のアナライザーを用いて測定した．測定に用いた計算パラメータは Table 4.1のものと同じである．2 つの表を比較して解ることは，3 次元 FFT の副プログラム（fft3d，fft3dx）を

Table 5.1 ベクトル最適化後の各副プログラムの負荷の全体に示す割合

Name	CPU time (sec)	Ratio (%)	V. op ratio (%)
fft3d	21.074	46.9	95.59
matset1	5.754	12.8	98.10
fft3dx	4.614	10.3	90.45
calchc	4.305	9.6	98.03
calcfec	4.174	9.3	98.28
mkvps	1.768	3.9	96.68
svxc	1.131	2.5	98.63
chgdns	0.622	1.4	97.97
ewald	0.615	1.4	82.73
reclat	0.233	0.5	0.18
mkbessel	0.125	0.3	34.29
nextwf	0.081	0.2	87.76
onchk	0.081	0.2	73.24

ベクトル最適化しものに差し替えたために，この副プログラムの処理にかかる時間の全体にしめる割合が大幅に減少したことである．結果として，並列化した部分の全体にしめる割合が大幅に減少してしまった．並列化した副プログラムは，fft3d，fft3dx，calchc，calcfec，mkvps，svxc で

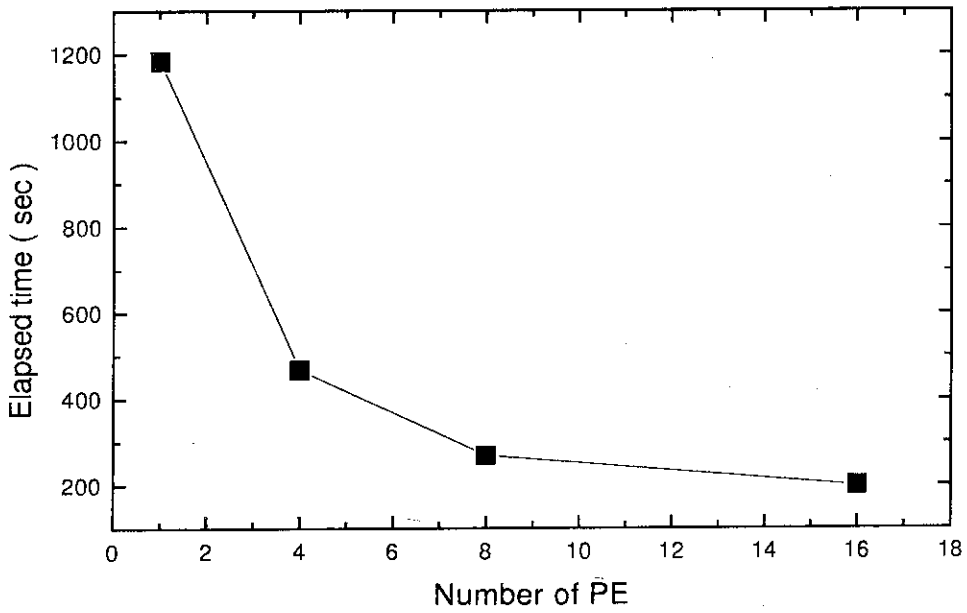

Fig. 5.1 ベクトル並列化した Camp-Atami による処理時間の PE 数依存性

あるから並列化した部分の全体にしめる割合は 83%程度である．もっとも，この値は分子動力学を行う系の特性（構成原子の種類，個数，計算に用いるエネルギーカットオフ等）によって幾分変化するのであくまで目安である．並列化されていない副プログラムで大きな割合をしめるのは，matset1 である（前章で述べたように matset1 から呼ばれるプログラムは並列化を行ったが本体については行っていない）．今後は並列化率を高めるために，副プログラム matset1 を並列化する必要がある．

次に，並列化による性能向上を見ていくことにする．Fig. 5.1に PE 数を増加させていったときの，計算時間の変化を示す．測定に用いた系は，単位格子に 8 個の Si 原子を含む結晶構造から各原子をすこしずらした系で，Camp-Atami にサンプルとして付属する入力データの一つである．カットオフエネルギーは 10Ry とし，原子を動かすステップを 50 回実行した．これはステップ数が少ない事を除けば，実際に物性研究を行う上で必要な計算に近いパラメータの選択である．これをスピードアップの観点からプロットしたものを，Fig. 5.2に示す．スピードアップはそれ程良いとはいえないが，これは前述のようにベクトル最適化を行った結果並列化した部分の割合が減少してしまったためである．しかしながら，スピードアップは 8PE までほぼプロセッサー数に比例して増加しており 8PE で 4.4 倍の性能を示す．それ以降は飽和する傾向にある．このプログラムの並列処理の実用域は 2-8PE であることがわかる．

図中の実線は，全体の 88%が並列化されたときの理論曲線（ Speed up = 1/(0.12+(0.88/PE 数) ）である．測定値はこの仮定により良く説明できることがわかる．Table 5.1から得られた並列化率よりも良くなっているが，これは計算する問題の規模を大きくして実際のケースに近づけたためである．大きな規模の計算（単位格子あたりより多くの原子を含む計算やエネルギーカッ

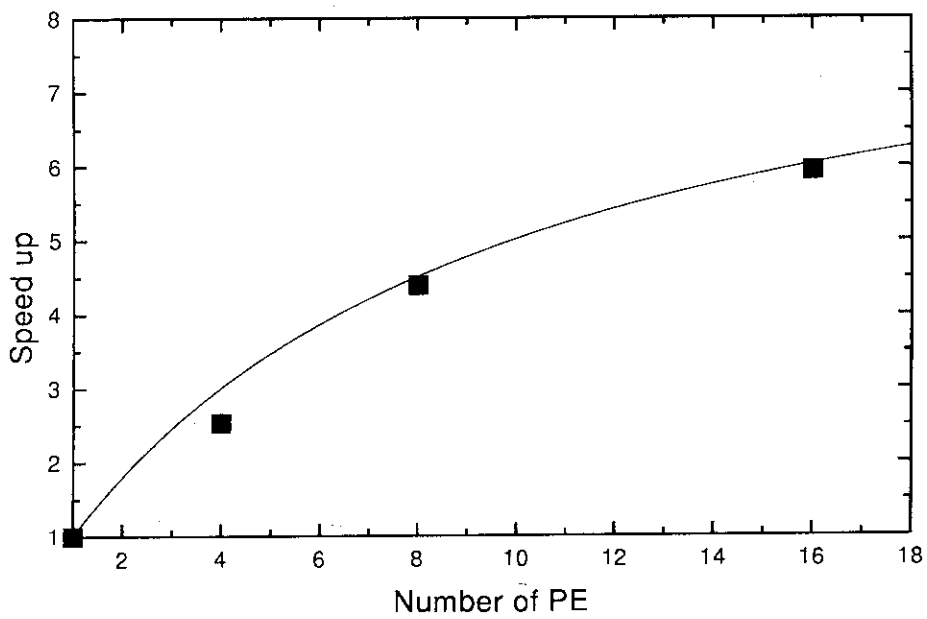

Fig. 5.2 ベクトル並列化した Camp-Atami のスピードアップ

トオフが大きい場合）では，より性能が向上すると思われる．

5.2 3 次元高速フーリエ変換の性能

ここでは，今回作成した 3 次元 FFT の性能評価を行う．比較したのは 3 次元 isogeometric FFT（以下 isogeometric FFT と呼ぶことにする），およびこれをベクトル最適化した改良版 isogeometric FFT（以下これを改良版 isogeometric FFT と呼ぶことにする）と SSL2 ライブラリーに含まれる並列版 FFT（以下これを SSL2 版 FFT と呼ぶことにする）の 3 つである．SSL2 版 FFT では高速フーリエ変換をおこなうデータが各 PE に分割されたままであるのに対し，今回作成したものは最後に重複ローカル変数にコピーして全ての PE で同じデータを持つようにしている．そこで，公平のため SSL2 版でも同じ処理を行って比較した．

Fig. 5.3に 32x32x32 の 3 次元データを上記 3 つの FFT を用いて高速フーリエ変換を行ったときの処理時間を PE 数に対してプロットしたものを示す．まず isogeometric FFT とその改良版を比べると，ベクトル最適化により 2-2.7 倍の高速化がはかられている事が解る．PE 数が増加するにつれ差が小さくなっているのは，後に述べるようにデータ転送にかかる時間が無視できなくなってくるからである．Camp-Atami にもともと付属の FFT と比較するとベクトル最適化を行った後では約 3.4 倍の高速化がはかれた．

次に今回作成した改良版 isogeometric FFT を SSL2 版 FFT と比較すると，1PE の時はわずかに劣るが，それ以外の PE 数では今回作成した FFT の方が高速である事が解る．特に SSL2 版 FFT は 8PE 以上では 1PE や 4PE の時に比べ，むしろ遅くなっている．ライブラリーの中でど

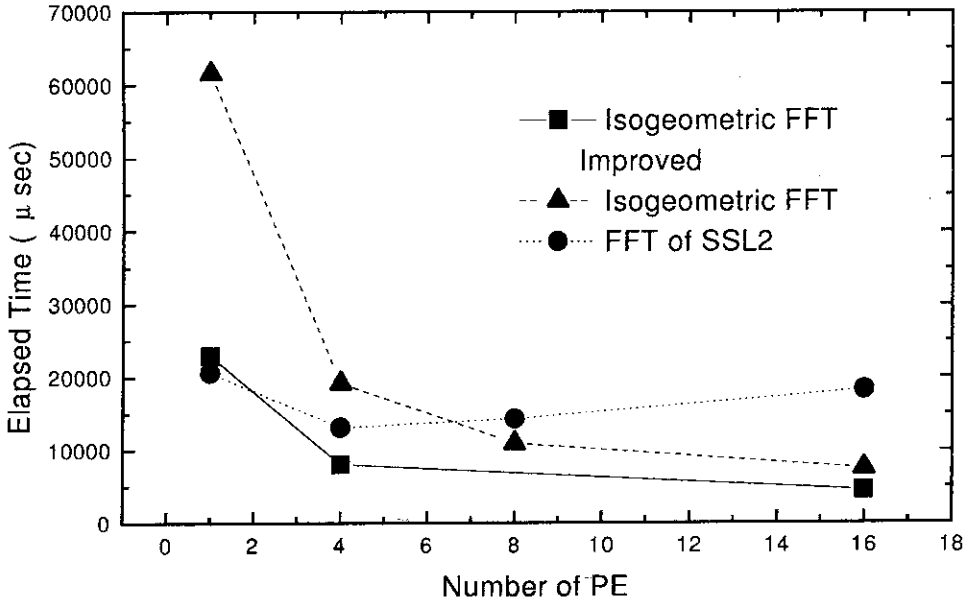

Fig. 5.3 並列版高速フーリエ変換の性能比較

の様な処理がされるかは公開されていないので，SSL2 版 FFT が PE 数を増やしたときに遅くなる理由ははっきりとは解らない．考えられる理由としては，(1) 並列化によりベクトル長が減少した，(2) データ転送の仕方あるいは転送量による特性，(3) バンク競合，等があげられる．そこで付録 1 で，幾つかの転送方法に対してその性能を評価した．またバンク競合をさける様に改良したプログラムを作成し，付録 2 でその性能を比較評価した．これらは並列化における他の場面でも役立つと思われる．

今度はスピードアップの観点から見ることにする．Fig. 5.4は 3 つの FFT のスピードアップを PE 数に対してプロットしたものである．SSL2 版 FFT は 8PE 以上でむしろ処理速度が低下するのに対応して，4PE 以上で減少し始める．一方，今回作成した FFT はスピードアップは PE 数とともに増加しているが飽和する傾向が見られる．これはデータ転送にかかる時間が無視できないからである．ベクトル最適化を行っていない FFT の方がスピードアップが良いのは，データの処理時間が長い分だけデータの転送時間の割合が低いからである．

データの転送にかかっている時間は PE 数によらずほぼ一定で 3000μ sec 位であった．これは PE 同士をクロスバー接続をしている VPP の特性と思われる．そこで isogeometric FFT とその改良版においてデータの転送にかかる時間や並列化のためオーバーヘッドなどの時間の総和が PE 数によらず一定で 3200μ sec として，それ以外の時間は PE 数に反比例すると仮定して引いた理論曲線が Fig. 5.4の 2 つの実線である．計測結果はこの仮定が正しいことを示している．

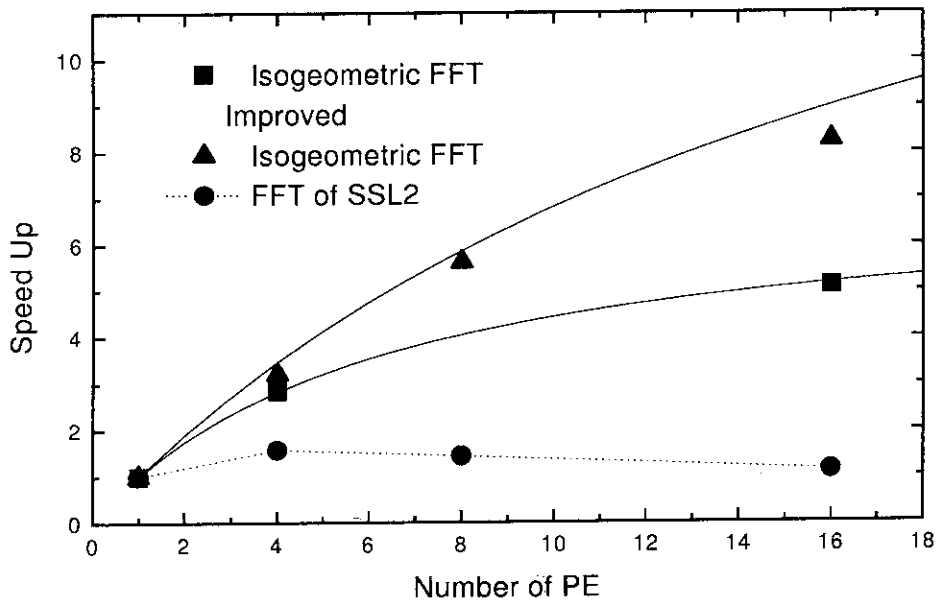

Fig. 5.4　並列版高速フーリエ変換におけるデータ分割の概念図

6. まとめと課題

日本化学計算プログラム交換機構（JCPE）を通して一般に配布されている第一原理分子動力学プログラム Camp-Atami をベクトル並列型スーパーコンピュータ VPP 500 用にベクトル並列化することを試みた．まずベクトル最適化により，単体で実行した時に 4.3 倍の高速化がはかれた．これには，FFT ルーティンを新たに作成しベクトル最適化を行ったことが大きく効いている．

次に，並列化によるプログラムのスピードアップは 8PE までほぼ PE 数に比例して伸び，8PE では 4.4 倍であった．倍率が低いのは，一番計算コストのかかっていた FFT ルーティンを前述の様に大幅に高速化したため，CPU 時間で見たときの FFT ルーティンの全体に占める負荷の割合が減少し，結果として並列化率が減少してしまったためである．並列化率は，計算する系の大きさ（原子数）やカットオフエネルギーに大きく依存し，計算規模が大きくなると並列化率は増加する．今回性能評価に用いた系は，分子動力学としては比較的小規模なものであるが，全体の 89%が並列化されているとしたときの理論曲線によってスピードアップは良く説明できる．

また，今回新たに並列版高速フーリエ変換を作成した．このプログラムは，2 基底のみという制限つきであるが VPP500 にライブラリーとして提供される並列版高速フーリエ変換プログラムに比べ，ほとんどの PE 数で高速である．FFT の並列化というと，流体計算などで使用されるような 1024x1024x1024 クラスの大規模な 3 次元データを効率良く計算するプログラムに目が行きがちで，32x32x32 程度の中規模な FFT に関してはチューニングがなされていない気がする．物性研究の場ではこうした中規模の FFT が重要になることも多いので一考を望む．

今回はもとのプログラムを物理的なスキームは変更せずに並列化するという立場で並列化を行ってきたが，物理的なスキームも含めて改良するという立場からはいくつかの改良点がある．

一つは並列化により，より多くのメモリー空間を取得しより大きな系での計算ができるようにするという事である．プログラム中で一番大きなメモリーを必要とするのは，場合により異なることもあるが，配列 hmat1，hmat2 であろう．これらは計算スキームの性格上，各 PE に分散して持つことができない．これらの配列は平面波の初期値を与えるために必要なわけであるが，初めは平面波の展開係数に乱数を与え，普通のカー・パリネロのスキームにしたがって電子系だけでクエンチを行って初期値を求める事で無くすことができる．

2 番めに大きなメモリーを必要とする配列は，平面波の展開係数を保存する cgjr(ig,ib)，cgji(ig,ib) やハミルトニアンの行列要素を保存する，hvxr(ig,ib)，hvxi(ig,ib)，hcgr(ig,ib)，hcgi(ig,ib)，rhcr(ig,ib)，rhci(ig,ib) などであるが，これらは各 PE ごとに分割して持つことができるので，より大きな系やあるいはより高いカットオフエネルギーでの計算が可能になる．

また，配列 bessel は擬ポテンシャルを計算するときに必要な式 (2.25) の結果を保存しているが，Kleinman と Bylander の提案した分離型擬ポテンシャル [7] を用いれば，式 (2.25) の計算は必要なくなり，同時に擬ポテンシャルの計算も各逆格子点について完全に並列に処理ができるようになる．

最後に，Camp-Atamiはもともとのカー・パリネロ法に非常に忠実につくられている．したがって，電子構造は波数空間全体ではなく原点（$k=0$）であるΓ点のみで計算されている．これは半導体や単純金属などフェルミエネルギー近傍の電子構造が比較的簡単なものでは問題ないが，複雑なものになってくるとより多くのk点での計算が必要になってくる．そのため複数k点での計算への拡張が考えられる．この場合，並列化の観点からは各k点については全く独立に計算できるため大変すっきりした大粒度の並列化が可能であり，並列化による高速化が期待できる．同時にこの場合はデータ分割も簡単に行え，並列化により大きなメモリーの取得も可能になる．並列化による処理時間の高速化と大きな系での計算を可能にするという2つの利点を実現できる理想的な計算となることを付け加えておく．

謝　　辞

ベクトル版の1次元Isogeometric高速フーリエ変換のソースコードを提供して頂いた，日本原子力研究所の横川研究員に感謝いたします．

参　考　文　献

[1] R. Car and M. Parrinello, Phys. Rev. Lett. **55**, 2471 (1985).

[2] 斉藤　隆之, 固体物理 **29**, 941 (1994).

[3] P. Hohenberg and W. Kohn, Phys. Rev. **136**, B864 (1964).

[4] W. Kohn and L.J. Sham, Phys. Rev. **140**, A1133 (1965).

[5] D.R. Hamann, M.Schlüter, and C. Ching, Phys. Rev. Lett. **43**, 1494 (1979).

[6] G.B. Bachelet, D.R. Hamann, and M. Schlüter, Phys. Rev. **26**, 4199 (1982).

[7] L. Kleinman and D.M. Bylander, Phys. Rev. Lett. **48**, 1425 (1982).

[8] P.P. Ewald, Ann. Physik **64**, 253 (1921).

[9] L. Verlet, Phys. Rev. **159**, 98 (1967).

[10] M. Yokokawa and H. Kaburaki, IPSJ SIG Notes HPC 46-7, pp.34-44 (1993) (in Japanese).

最後に，Camp-Atamiはもともとのカー・パリネロ法に非常に忠実につくられている．したがって，電子構造は波数空間全体ではなく原点（$\boldsymbol{k}=0$）であるΓ点のみで計算されている．これは半導体や単純金属などフェルミエネルギー近傍の電子構造が比較的簡単なものでは問題ないが，複雑なものになってくるとより多くのk点での計算が必要になってくる．そのため複数k点での計算への拡張が考えられる．この場合，並列化の観点からは各k点については全く独立に計算できるため大変すっきりした大粒度の並列化が可能であり，並列化による高速化が期待できる．同時にこの場合はデータ分割も簡単に行え，並列化により大きなメモリーの取得も可能になる．並列化による処理時間の高速化と大きな系での計算を可能にするという２つの利点を実現できる理想的な計算となることを付け加えておく．

謝　　辞

ベクトル版の１次元 Isogeometric 高速フーリエ変換のソースコードを提供して頂いた，日本原子力研究所の横川研究員に感謝いたします．

参　考　文　献

[1] R. Car and M. Parrinello, Phys. Rev. Lett. **55**, 2471 (1985).

[2] 斉藤　隆之, 固体物理 **29**, 941 (1994).

[3] P. Hohenberg and W. Kohn, Phys. Rev. **136**, B864 (1964).

[4] W. Kohn and L.J. Sham, Phys. Rev. **140**, A1133 (1965).

[5] D.R. Hamann, M.Schlüter, and C. Ching, Phys. Rev. Lett. **43**, 1494 (1979).

[6] G.B. Bachelet, D.R. Hamann, and M. Schlüter, Phys. Rev. **26**, 4199 (1982).

[7] L. Kleinman and D.M. Bylander, Phys. Rev. Lett. **48**, 1425 (1982).

[8] P.P. Ewald, Ann. Physik **64**, 253 (1921).

[9] L. Verlet, Phys. Rev. **159**, 98 (1967).

[10] M. Yokokawa and H. Kaburaki, IPSJ SIG Notes HPC 46-7, pp.34-44 (1993) (in Japanese).

最後に，Camp-Atamiはもともとのカー・パリネロ法に非常に忠実につくられている．したがって，電子構造は波数空間全体ではなく原点（$k=0$）であるΓ点のみで計算されている．これは半導体や単純金属などフェルミエネルギー近傍の電子構造が比較的簡単なものでは問題ないが，複雑なものになってくるとより多くのk点での計算が必要になってくる．そのため複数k点での計算への拡張が考えられる．この場合，並列化の観点からは各k点については全く独立に計算できるため大変すっきりした大粒度の並列化が可能であり，並列化による高速化が期待できる．同時にこの場合はデータ分割も簡単に行え，並列化により大きなメモリーの取得も可能になる．並列化による処理時間の高速化と大きな系での計算を可能にするという2つの利点を実現できる理想的な計算となることを付け加えておく．

謝　　辞

ベクトル版の1次元Isogeometric高速フーリエ変換のソースコードを提供して頂いた，日本原子力研究所の横川研究員に感謝いたします．

参　考　文　献

[1] R. Car and M. Parrinello, Phys. Rev. Lett. **55**, 2471 (1985).

[2] 斉藤　隆之, 固体物理 **29**, 941 (1994).

[3] P. Hohenberg and W. Kohn, Phys. Rev. **136**, B864 (1964).

[4] W. Kohn and L.J. Sham, Phys. Rev. **140**, A1133 (1965).

[5] D.R. Hamann, M.Schlüter, and C. Ching, Phys. Rev. Lett. **43**, 1494 (1979).

[6] G.B. Bachelet, D.R. Hamann, and M. Schlüter, Phys. Rev. **26**, 4199 (1982).

[7] L. Kleinman and D.M. Bylander, Phys. Rev. Lett. **48**, 1425 (1982).

[8] P.P. Ewald, Ann. Physik **64**, 253 (1921).

[9] L. Verlet, Phys. Rev. **159**, 98 (1967).

[10] M. Yokokawa and H. Kaburaki, IPSJ SIG Notes HPC 46-7, pp.34-44 (1993) (in Japanese).

付録1 VPP 上でのデータ統合と転置

VPP 上でプログラムの並列化を行っていると，しばしば各 PE で持っている重複ローカルデータを合計して全ての PE で同じデータを持つようにする必要が生じる．これを Fujitsu ではデータの統合（unify）と呼んでいる．しかし，場合によってはデータを分割ローカル変数として持ち，データの分割の仕方を変えて転送する（転置を行う）だけですむ場合もある．一般にデータ転置はデータ統合に比べ高速であると言われるが具体的なデータはあまりない．そこで，VPP におけるデータ統合（spread sum）とデータ転置（spread move）の性能を以下の3つの観点から比較検討してみた．

- 配列形状に対する依存性
- PE 数依存性
- 転置の仕方による依存性

これらを調べるために，以下のモデルを設定した．まず，配列形状依存性を調べるために要素数は 16384（2^{14}）と同じで切り方が，1024x16，512x32，256x64，128x128，64x256，16x1024 と異なる配列を用意した．以下では，簡単のために A(1024,16/4) というような記号を使うことにする．これは配列 A が 1024x16 の大きさの配列で2次元目が4つの PE に分割されている分割ローカル変数である事を示す．

また，統合および転置の仕方については次の5つのモデルタイプを考えた．

i. 2次元重複ローカル変数 $A(nx, ny)$ を全ての PE にわたって合計し，それを全ての PE に配りなおす．これを，
$A(nx, ny) \leftarrow A(nx, ny)$
で表すことにする．

ii. y 軸を各 PE で分割して持っている2次元分割ローカル変数 $B(nx, ny/PE)$ を，x 軸を分割したローカル変数 $C(nx/PE, ny)$ に転置する．これを，
$C(nx/PE, ny) \leftarrow B(nx, ny/PE)$
で表すことにする．

iii. x 軸を各 PE で分割して持っている2次元分割ローカル変数 $C(nx/PE, ny)$ を，y 軸を分割したローカル変数 $B(nx, ny/PE)$ に転置する．これを，
$B(nx, ny/PE) \leftarrow C(nx/PE, ny)$
で表すことにする．

iv. y 軸を各 PE で分割して持っている2次元分割ローカル変数 $B(nx, ny/PE)$ を，x 軸を分割したローカル変数 $tC(ny, nx/PE)$ に転置する．2番目の操作とほぼ同じであるが，行と列

の転置も行う点が異なる．これを，
$tC(ny, nx/PE \leftarrow B(nx, ny/PE)$
で表すことにする．

v. x 軸を各 PE で分割して持っている 2 次元分割ローカル変数 $C(nx/PE, ny)$ を，y 軸を分割したローカル変数 $tB(ny/PE, nx)$ に転置する．3 番目の操作とほぼ同じであるが，行と列の転置も行う点が異なる．これを，
$tB(ny/PE, ny) \leftarrow C(nx/PE, ny)$
で表すことにする．

上に述べた 5 つの形状の異なる配列に対し，これら 5 つのデータ転送を PE 数を変えながら行ってみた．その中で顕著な違いがあったものを紹介する．Fig. A.1に配列の切り方は 512x32 で同じデータに対して，転送方式と PE 数を変化させたときの転送時間を示す．一番左の棒グラフは，重

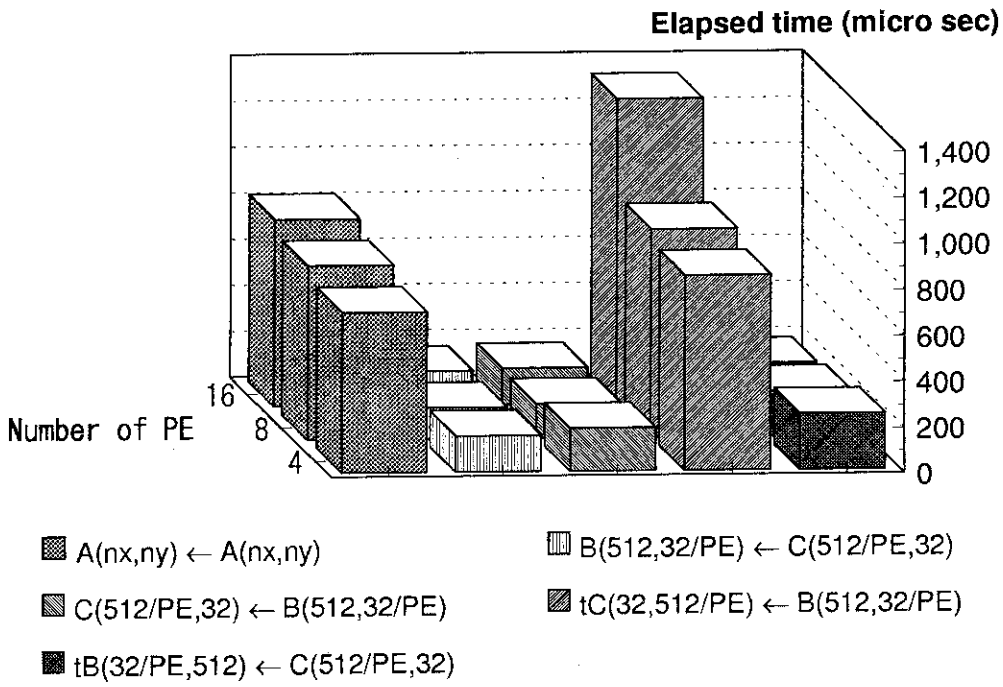

Fig. A.1 データ転送時間の転送方法及び PE 数依存性

複ローカルデータの統合をしたときの PE 数依存性であるが，PE 数にはほとんど依存せず一定であることが解る．また，$tC(ny, nx/PE) \leftarrow B(nx, ny/PE)$ 型の転置以外の転置は統合に比べかなり速く PE 数依存性もあまりないことが解る．一方，$tC(ny, nx/PE) \leftarrow B(nx, ny/PE)$ 型の転置は他の転置に比べ非常に遅く，また PE 数依存性も強い．データ統合に比べ遅いこともある．従って，このタイプのデータ転置が必要なときは，先ず $C(nx/PE, ny) \leftarrow B(nx, ny/PE)$ 型の転置を行っておいて各 PE でローカル変数同士の転置 $tC(ny, nx/PE) \leftarrow C(nx/PE, ny)$ を行ったほう

が速い．こうした転置は，ベクトル最適化においてベクトル長を長くするために必要となる．

次に，$tC(ny, nx/PE) \leftarrow B(nx, ny/PE)$ 型の転置の配列形状依存性と PE 数依存性を Fig. A.2 に示す．このタイプのデータ転置では，配列形状依存性も大きいことが解る．以上，見てきたように VPP でのデータ転送はかなり癖があるので並列化においては注意が必要である．

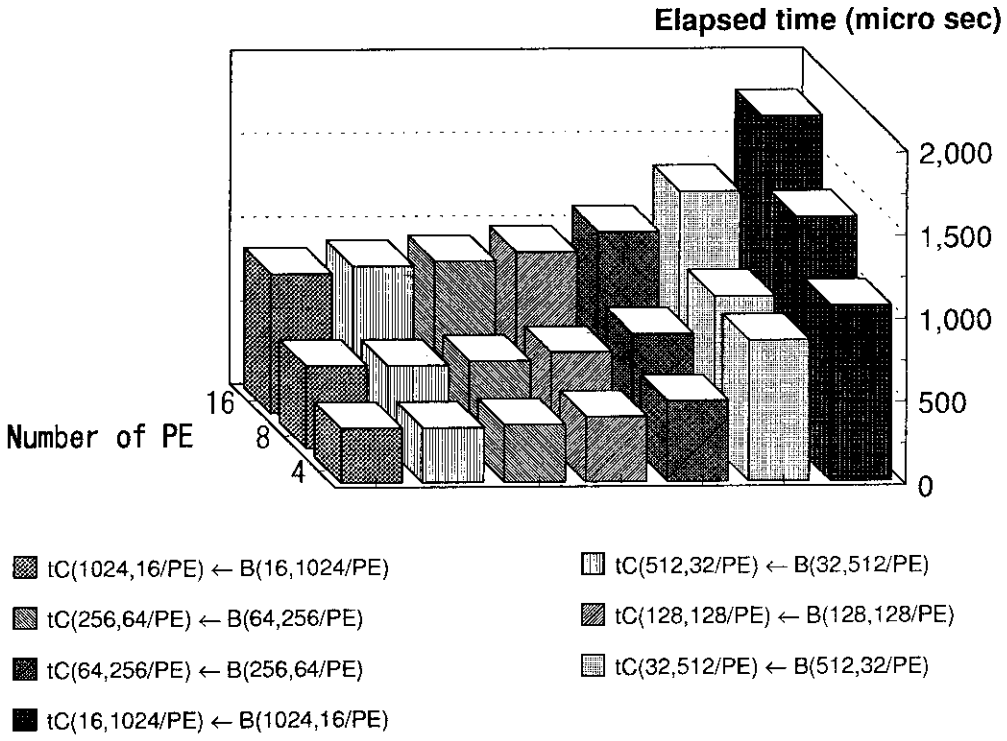

Fig. A.2 データ転送時間の配列形状及び PE 数依存性

付録2 バンク競合

今回作成した3次元FFTは，2基底の3次元配列を扱うことからバンク競合が発生する．そこで，4.4節で紹介したisogeometric FFTを改良して，1次元目の配列の大きさを2^n+1とすることでバンク競合をさけるプログラムを作成した．Fig. A.3に，第4.4節で紹介したisogeometric FFT，これを改良してバンク競合を取り除いたFFT，SSL2ライブラリーのFFTの3つのFFTプログラムで32x32x32の大きさのデータをフーリエ変換したときの処理時間のPE数依存性を示す．図からバンク競合の影響は非常に大きく，単体実行したときには，バンク競合をさけることで倍以上の高速化がはかれる事がわかる．しかしながら，バンク競合を取り除いたFFTでは，8PEまでは処理時間は減少するがそれ以降処理時間はむしろ増加する．この処理時間の増加は，バンク競合をさけるために，データを保存する配列の大きさを33x32x32としたことで，32x32x32の時に比べ転送の際に余分なパケット転送が生じたと考えるのが妥当と思われる．

PE数を増加したときに処理時間が途中から増加に転じる様子は，SSL2ライブラリーのFFTの挙動と良く似ており，同じ理由によるものかもしれない．しかし，前述の様にSSL2ライブラリーは中での処理は公開されていないので，予測の域をでない．

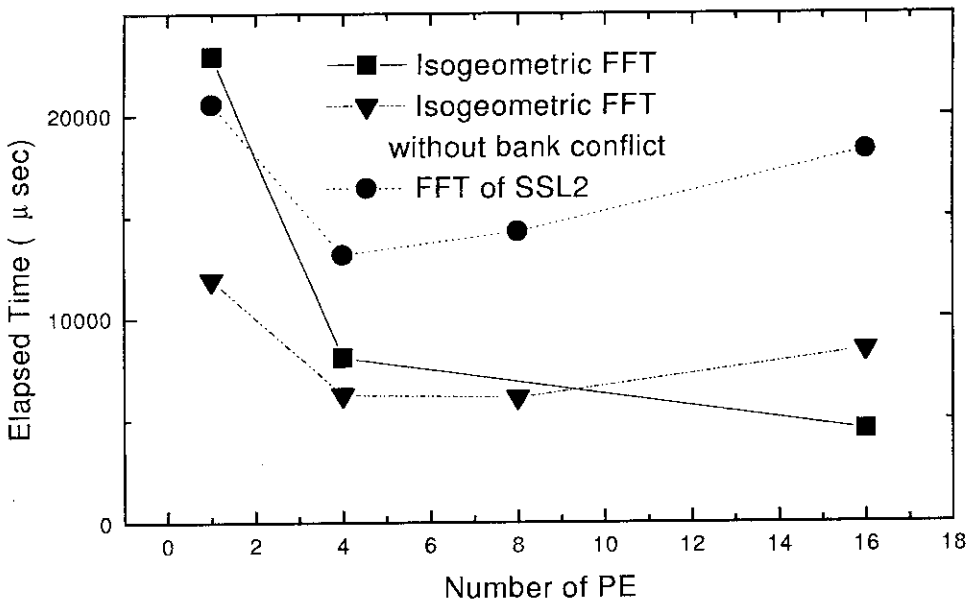

Fig. A.3 各種FFTの処理時間のPE数依存性の比較

付録3 プログラムのコンパイルと実行

今回並列化したプログラムを，原子力研究所の VPP500 上で使用する方法を簡単に紹介する．プログラムは，tar コマンドにより1つのファイルにまとめられている．これを解凍するためには，適当なディレクトリーで，

```
tar xvf camp.tar
```

とすれば，ファイルが解凍され camp というディレクトリーの下に，src と test というディレクトリーが作成される．ディレクトリー src の下には，ソースファイルが置かれ，test のしたにはサンプル入力ファイルが幾つかつくられる．

次にメイクの仕方を述べる．メイクをする時には，まず幾つの PE で実行するかを決める．PE 数は，ディレクトリー src の下にある size.inc の中の parameter 文で'npe=' の後に指定する．次に，メイクをするためには，ディレクトリー camp で'MAKE' とするだけで良い．途中コンパイルオプションを聞かれる（Fig. A.4参照）．作成したい実行ファイルが，スカラー単体実行用であれば'0' を，スカラー並列実行用であれば'1' を，ベクトル単体実行用であれば'2' を，ベクトル並列実行用であれば'3' をそれぞれ入力すれば良い．但し，size.inc で指定した PE 数と矛盾しないようにする．これで，実行ファイル'camp' がディレクトリー camp の下のディレクトリー test に作成される．

プログラムの実行には，入力ファイルとして carparr.dat と bhspp.dat が必要である．前者は分子動力学を行う系や計算条件を指定する．詳しくは，Camp-Atami に付属のマニュアルを読んでいただきたい．サンプルとして，単位胞に8個の Si 原子を結晶配置から少しずらして置き，3回原子位置を動かす計算の入力ファイルを用意した．ファイル bhspp.dat は，全ての原子に対する擬ポテンシャルのパラメータが記述されていて，これは変更する必要はない．これらのファイルは，ディレクトリー test の下にある．

最後に，ジョブを投入するシェルスクリプトの例を Fig. A.5に示す．特に他のプログラムと変わるところはない．投入するジョブクラスと PE 数は，size.inc で指定したものと矛盾しないようにする．

```
Please specify optimization level.
+-LEVEL-----------------------------+
|   0    scalar serial              |
|   1    scalar parallel            |
|   2    vector serial              |
|   3    vector parallel            |
|   d    for debugger               |
|   p    for profiler               |
+-----------------------------------+
Type LEVEL=
```

Fig. A.4　メイク時に指定できるコンパイルオプション

```
#!/bin/csh -f
#@$-C md
#@$-q vpp4
#@$-eo
#@$-lPv 1
cd ~/campvpp/test
setenv fu10 ./carparr.dat
setenv fu11 ./fort.11
setenv fu12 ./fort.12
setenv fu13 ./fort.13
setenv fu14 ./fort.14
setenv fu15 ./fort.15
setenv fu16 ./fort.16
setenv fu17 ./fort.17
setenv fu21 ./fort.21
setenv fu22 ./fort.22
setenv fu23 ./fort.23
setenv fu31 ./fort.31
setenv fu32 ./fort.32
setenv fu33 ./fort.33
setenv fu34 ./fort.34
setenv fu35 ./fort.35
setenv fu36 ./fort.36
setenv fu37 ./fort.37
setenv fu41 ./fort.41
timex -H ./camp -Wl,-Pg1024
```

Fig. A.5 ジョブスクリプトの例