

JAERI-Data/Code
97-009



第一原理電子状態計算プログラムの並列化

1997年3月

渡部 弘・小口多美夫*

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、日本原子力研究所研究情報部研究情報課（〒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 Research Information Division, Department of Intellectual Resources, Japan Atomic Energy Research Institute, Tokai-mura, Naka-gun, Ibaraki-ken 319-11, Japan.

© Japan Atomic Energy Research Institute, 1997

編集兼発行 日本原子力研究所
印 刷 株原子力資料サービス

第一原理電子状態計算プログラムの並列化

日本原子力研究所計算科学技術推進センター

渡部 弘・小口多美夫*

(1997年2月10日受理)

第一原理電子状態計算プログラムの並列化について報告する。ターゲットマシンは共有メモリ型マシンにNEC SX-4、分散メモリ型マシンに富士通VPP300を採用した。各々のマシンの特徴について概観した後、各々の特性を生かした並列化を行った。その結果、SX-4では2並列で1.60倍の加速率が、VPP300では12並列で最大4.97倍の加速率が得られた。

Parallelization for First Principles Electronic State Calculation Program

Hiroshi WATANABE and Tamio OGUCHI*

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

(Received February 10, 1997)

In this report we study the parallelization for First principles electronic state calculation program. The target machines are NEC SX-4 for shared memory type parallelization and FUJITSU VPP300 for distributed memory type parallelization. The features of each parallel machine are surveyed, and the parallelization methods suitable for each are proposed. It is shown that 1.60 times acceleration is achieved with 2 CPU parallelization by SX-4 and 4.97 times acceleration is achieved with 12 PE parallelization by VPP 300.

Keywords: First Principles, Electronic State, Parallelization, SX-4, VPP300

* Hiroshima University

目 次

1. はじめに	1
2. FLAPW 法	2
3. 計算コスト調査	3
4. 並列化方針	5
5. SX-4 での並列化	7
5.1 SX-4 の特徴	7
5.2 並列化方法	7
5.3 性能測定結果	10
5.4 考 察	11
6. VPP300 での並列化	12
6.1 VPP300 の特徴	12
6.2 UNIFY 手法	12
6.3 並列化方法	14
6.4 性能測定結果	16
6.5 考 察	18
7. まとめ	20
謝 辞	20
参考文献	20
付 錄	21

Contents

1. Introduction	1
2. FLAPW Method	2
3. Investigation of Computing Cost	3
4. Parallelization Strategy	5
5. Parallelization on SX-4	7
5.1 Features of SX-4	7
5.2 Parallelization Method	7
5.3 Performance Evaluation	10
5.4 Discussion	11
6. Parallelization on VPP300	12
6.1 Features of VPP300	12
6.2 UNIFY Method	12
6.3 Parallelization Method	14
6.4 Performance Evaluation	16
6.5 Discussion	18
7. Conclusion	20
Acknowledgements	20
References	20
Appendix	21

1 はじめに

固体中の電子状態計算は、物性研究や各種新物質、新材料開発にとって非常に重要である。特に密度汎関数理論に基づく電子状態計算は、調整可能なパラメータを含まないという非経験性のゆえに、第一原理計算として発展しており、多くの信頼性の高い結果を与えていた[1]。

しかし一方、必要な計算リソースは計算時間、記憶容量とともに膨大であり、現時点でのパソコンコンピュータやワークステーションで実用的な計算を行うには多大な困難が伴う。従ってスーパーコンピュータの利用が不可欠なのだが、従来のベクトル型スーパーコンピュータではやはり荷が重い。たとえば本報告書で説明している FLAPW 法による第一原理計算を従来のベクトル型スーパーコンピュータで計算すると、妥当な結果が得られるまでに約 12 時間の CPU 時間を要する。このことはセンターマシンのような多人数で共用する状態では、結果を得るまでにより多くの経過時間 (elapse time) が必要ということであり (多くの場合 CPU 時間の 2 倍以上)、研究現場および設計現場で頻繁に行われる trial and error に重大な支障をきたす。そこでより高速な第一原理計算プログラムの開発は急務といえる。

以上のような背景のもと、我々は FLAPW (Full potential Linear Augmented Plane Wave) 法による第一原理電子状態計算プログラムの並列化を行った。単一 CPU による性能向上が頭打ちになっている今日、並列化による高速化は唯一の解決策といえる。幸い本プログラムは原理的に高粒度の並列性を有しており、かなりの高速化が期待できる。

並列マシンはそのメモリ実装方式の違いにより、共有メモリ型と分散メモリ型に分類される。共有メモリ型は 1 つの大容量のメモリを複数個の CPU が共有しているのに対し、分散メモリ型は各々の CPU が固有のメモリを所有している。各々の方式には一長一短があり、どちらが優れているとは一概には言えない。並列化手法も共有メモリ型と分散メモリ型では大幅に異なる。本研究ではできるだけ多くの並列機をサポートするという観点から、両方式に対して別々の並列化を試みた。ターゲットマシンとしては共有メモリ型並列マシンとして NEC SX-4、分散メモリ型並列マシンとして富士通 VPP300 を採用した。両マシンとも日本原子力研究所 計算科学技術推進センター (以下 当センターと略す) が所有するマシンである。

本報告書の構成は以下の通りである。第 2 章は本研究で並列化される第一原理電子状態計算プログラムの原理となる FLAPW 法について簡単に説明する。第 3 章では FLAPW 法プログラムを非並列で実行することにより計算時間コスト解析を行い、並列化すべきルーチンを抽出する。第 4 章ではそれらのルーチンに対する原理的な並列化方針について記述する。第 5 章と第 6 章では、第 4 章の並列化方針に従って SX-4 及び VPP300 上でプログラムを並列化し、並列化プログラムの性能を測定する。

並列計算の現状を省みると、残念ながら並列化プログラミングがエンドユーザに十分に浸透しているとは言い難い。その理由はいろいろ考えられるが、プログラムの並列化技術がベクトル化技術等に比べ格段に高度かつ困難であるにも関わらず、あるいはそうであるからこそ、プログラマをサポートする技術(たとえば自動並列化コンパイラや各種並列化ツール) がいまだ未熟であることに起因すると思われる。もちろんこれらの技術は着実に進展しており多大な成果も出ているのであるが、ここ当面はプログラマ自身の”人手による” 並列化は避けられない状況が続くであろう。このような状況下で重要なのは、種々のプログラムを実際に並列化することによって得られる具体的な手法やノウハウの蓄積であり、その普及のための地道な努力である。そのため本報告書では、第 5 章及び第 6 章で共有メモリ型並列マシン及び分散メモリ型並列マシンでのプログラム並列化について詳しく述べた。本報告書がこれから並列化を行おうとする多くの第一線で活躍されている研究者の一助となるならば、筆者のこれに過ぎる喜びはない。

2 FLAPW 法

FLAPW(Full potential Linear Augmented Plane Wave) 法 [2] は Krakauer 等により 1979 年に提案された。この手法は密度汎関数理論にその基礎を置いており、以下の (Rydberg 原子単位による)Kohn-Sham 方程式を局所密度近似のもとで、セルフコンシステントに解を求める手法である。

$$[-\nabla^2 + v_{eff}(\mathbf{r})] \psi_i(\mathbf{r}) = \varepsilon_i \psi_i(\mathbf{r}) \quad (1)$$

v_{eff} は有効ポテンシャルと呼ばれ、注目している一電子が結晶内で感じるポテンシャルである。 ε_i, ψ_i は各々、エネルギーと波動関数である。本来、対象とする系には相互作用しあう多くの電子が含まれ(多電子系) 厳密な取り扱いが困難である。しかしながら、局所密度近似の導入によりこの有効ポテンシャル中の相互作用のない一体の問題に帰着され、解法が可能となる。

有効ポテンシャル $v_{eff}(\mathbf{r})$ は各原子核の電荷 ($+eZ$) によるポテンシャル $v_{nucleus}(\mathbf{r})$ 、電子密度

$$n(\mathbf{r}) = \sum_i |\psi_i(\mathbf{r})|^2 \quad (2)$$

による静電的なポテンシャル $v_{electron}(\mathbf{r})$ 、および局所密度近似により表現された $n(\mathbf{r})$ を含む交換相関ポテンシャル $\mu_{xc}(\mathbf{r})$ の和として表わされる。

$$v_{eff}(\mathbf{r}) = v_{nucleus}(\mathbf{r}) + v_{electron}(\mathbf{r}) + \mu_{xc}(\mathbf{r}) \quad (3)$$

これらの式は、密度汎関数理論により与えられた電子密度の汎関数としての電子系の全エネルギーの密度に関する変分原理から導かれたものである。

このことにより、固体結晶系内部の電子状態は原理的には図 1 のアルゴリズムで計算できる。

一般に固有値問題には大きく分けて、行列を陽に対角化する方法とレイリー商を最小化する方法の 2 種類の解法がある。本研究では必要なメモリが少ないと、並列化が容易なことからレイリー商を最小化する方法を採用している。従って Kohn-Sham 方程式 (1) は反復法で解かれており、その反復と区別するために、前回のセルフコンシステントな解を得るための反復を外部反復と呼ぶことにする。

Kohn-Sham 方程式 (1) を数値的に解くために波動関数 ψ_i を適当な基底関数で展開する。

FLAPW 法では、各原子 α の周りに muffin-tin 球とよばれる球(半径 s_α)を仮定し、その球内領域では球面波を用いて平面波関数を補強(augment)する基底関数を採用する。具体的には球面波 $\phi_{\alpha\ell m}(\mathbf{r}; E)$ を用いて次式のように書ける。

$$\phi_n(\mathbf{r}) = \tilde{\phi}_n(\mathbf{r}) + \sum_{\alpha} \Theta(s_{\alpha} - r_{\alpha}) \left[\sum_{\ell m}^{\ell_{max}} \phi_{\alpha\ell m n}(\mathbf{r}_{\alpha}) - \sum_{\ell m}^{\ell'_{max}} \tilde{\phi}_{\alpha\ell m n}(\mathbf{r}_{\alpha}) \right] \quad (4)$$

ここで、 $\mathbf{r}_{\alpha} = \mathbf{r} - \mathbf{R}_{\alpha}$ で、 \mathbf{R}_{α} は原子 α の位置座標ベクトルである。また ℓm に関する和の記号として、 $\sum_{\ell m}^{\ell_{max}} = \sum_{\ell=0}^{\ell_{max}} \sum_{m=-\ell}^{\ell}$ と略記した。式 (4) の第 1 項は平面波関数で次の様に与えられる。

$$\tilde{\phi}_n(\mathbf{r}) = \Omega^{-1/2} \exp(i\mathbf{k}_n \cdot \mathbf{r}). \quad (5)$$

ここで、 $\mathbf{k}_n = \mathbf{k} - \mathbf{K}_n$ で、 \mathbf{K}_n は n 番目の逆格子ベクトルである。また式 (4) の第 2 項において、 $\Theta(x)$ はヘビサイドステップ関数である。

Kohn-Sham 方程式 (1) に課される境界条件は、結晶中の状態を区別する波数 k により異なる周期境界条件となる。 k の値は通常 10 から 100 の間である。各々の方程式は独立に解くことができ、このことがプログラムの効率的な並列化を可能にしている。

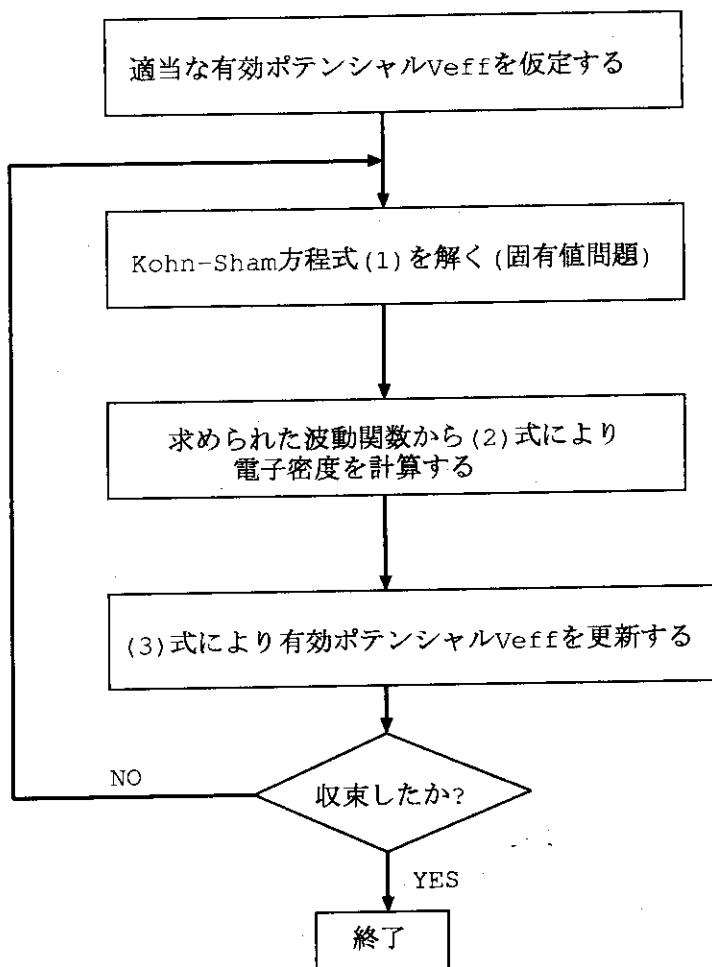


図 1: 第一原理電子状態計算のアルゴリズム

3 計算コスト調査

はじめに本プログラムのベクトル化状況調査および高コストルーチン切り出しのために、簡単なテスト用データを入力データとして、NEC SX-4でベクトル化率および各ルーチンの CPU TIME を測定した。テスト用計算としては、タンクステン表面に酸素を吸着させた場合の結晶構造と電子状態を解析するためのデータを用い、本来 50~100 回反復させる計算を 1 回に制限して行った。

SX-4にはプログラムの性能を落とすことなくプログラム性能情報を出力する機能がサポートされており、プログラム全体のベクトル化率や FLOPS 値を非常に容易に求めることができる。この機能を用いて本プログラムの性能を占有状態で計測した。(プログラム性能情報参照)

この情報によるとベクトル化率は 97%以上なので、さらに手動でループにベクトル化を施すことによる高速化は期待できないことがわかる。また占有状態での計測では本来、Real Time = User Time + Sys Time が成立するはずであるが、上記データでは 261 秒程度のオーバーヘッドがある。この大部分はディスクへの入出力のための時間である。従って高速な入出力デバイスが存在するならば、それを積極的に利用すべきである。

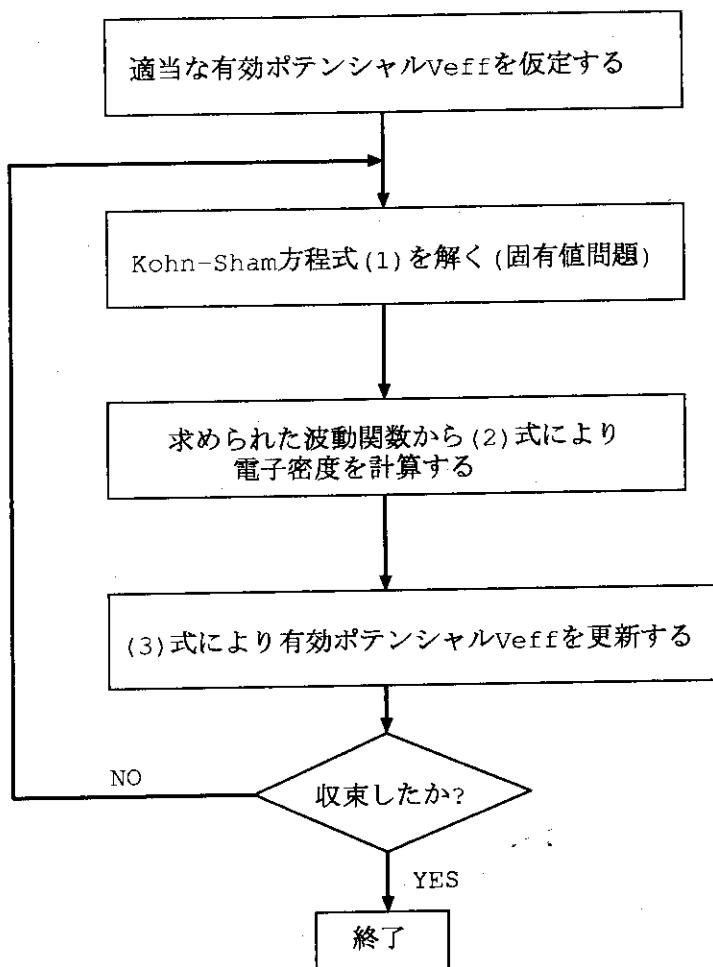


図 1: 第一原理電子状態計算のアルゴリズム

3 計算コスト調査

はじめに本プログラムのベクトル化状況調査および高コストルーチン切り出しのために、簡単なテスト用データを入力データとして、NEC SX-4でベクトル化率および各ルーチンの CPU TIME を測定した。テスト用計算としては、タンクステン表面に酸素を吸着させた場合の結晶構造と電子状態を解析するためのデータを用い、本来 50~100 回反復させる計算を 1 回に制限して行った。

SX-4 にはプログラムの性能を落とすことなくプログラム性能情報を出力する機能がサポートされており、プログラム全体のベクトル化率や FLOPS 値を非常に容易に求めることができる。この機能を用いて本プログラムの性能を占有状態で計測した。(プログラム性能情報参照)

この情報によるとベクトル化率は 97% 以上なので、さらに手動でループにベクトル化を施すことによる高速化は期待できないことがわかる。また占有状態での計測では本来、Real Time = User Time + Sys Time が成立するはずであるが、上記データでは 261 秒程度のオーバーヘッドがある。この大部分はディスクへの入出力のための時間である。従って高速な入出力デバイスが存在するならば、それを積極的に利用すべきである。

***** Program Information *****

Real Time (sec) : 811.997482
 User Time (sec) : 520.957229
 Sys Time (sec) : 31.546543
 Vector Time (sec) : 402.622795
 Inst. Count : 23340755295.
 V. Inst. Count : 6309408601.
 V. Element Count : 685586805858.
 FLOP Count : 297551253693.
 MOPS : 1348.706023
 MFLOPS : 571.162539 ← FLOPS 値
 VLEN : 108.661025
 V. Op. Ratio (%) : 97.576017 ← ベクトル化率
 Memory Size (MB) : 197.031250 ← メモリサイズ
 MIPS : 44.803592
 I-Cache (sec) : 9.428870
 O-Cache (sec) : 9.218893
 Bank (sec) : 56.692155

プログラム性能情報

次にサブルーチンの前後に CPU TIME 計測システム関数を挿入し、サブルーチンに費やされている CPU TIME を計測した。その結果、上位 3 ルーチンで全体の 78.9% の計算時間が消費されていることが判明した。(表 1 参照)

ルーチン名	cpu time (sec)	全体に占める比率 (%)
TOTAL	524.7	100.0
charsw	173.3	33.0
forcex	166.0	31.6
orthon	74.5	14.2

表 1: 主要ルーチンの時間コスト

ここで、charsw は球面波における電荷計算、forcex はエネルギー汎関数の波動関数による微分計算、orthon は計算で得られた波動関数の正規直交化をそれぞれ行っている部分である。これらは外部反復内の計算であり、テスト用例題では 1 回限りの計算であるが、本来は 50 回以上繰り返されるべきものである。従って上記の 78.9% という値は実用データではより大きな値になるはずであり、上記 3 ルーチンの並列化に注力すればよいことが予想される。

4 並列化方針

第2章で述べた計算時間コストの高い上位3ルーチンのソースコードを付録に掲載する。注目すべきは全てのルーチンが $\text{do } 20 \text{ k} = 1, \text{nkpt}$ という粒度の大きい最外側ループを有することである。このループは第1章で述べたように、結晶中の状態を区別する波数 k に関するループであり、Kohn-Sham 方程式(1)を異なる周期境界条件で解くことに由来する。この方程式は独立に解くことができ、従って $\text{do } 20 \text{ k} = 1, \text{nkpt}$ は原理的に並列化が可能なループである。そこで本研究では上記3ルーチンの $\text{do } 20 \text{ k} = 1, \text{nkpt}$ ループの並列化を中心に行い、他のルーチンにはできるだけ変更を加えないようとする。なぜなら後で述べるようにこのルーチンを並列化するだけで十分な性能向上が見込める上、必要以上のソース変更は予期せぬ精度低下やバグの混入を招きかねないためである。

nkpt は通常 10 以上 100 以下の値であり、またループ内の計算密度が高いことを考えると、CPU 数は比較的の少數でも 1CPU の性能の高い並列ベクトルマシンがターゲットマシンとして有望である。このような観点から、共有メモリ型並列マシンに NEC SX-4 を、分散メモリ型並列マシンに富士通 VPP300 をターゲットマシンに採用した。

理想的に並列化された場合の性能を見積もると次のようなになる。並列化率(プログラムの全 CPU TIME における並列化可能部分の CPU TIME の比率)を R_p とおくと、プロセッサ数 N_p の時の加速率 P は次式で予測できる。

$$P = \frac{1}{1 - R_p + R_p/N_p} \quad (6)$$

本プログラムでは $R_p = 0.789$ なので、理想的には図2のような高速化がなされる。

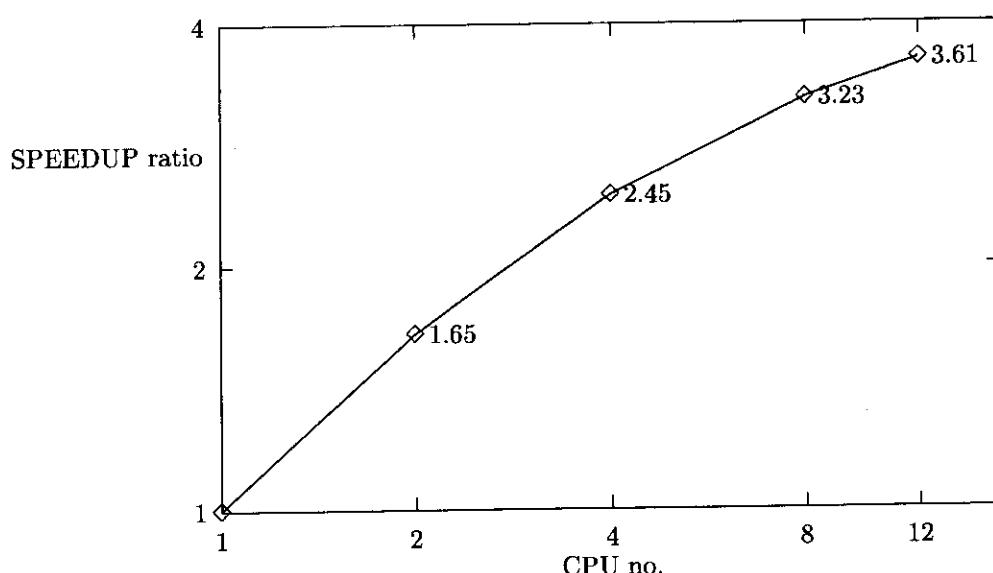


図2: CPU 台数と加速率の関係

原理的に並列化可能とはいえ、実際にプログラムを並列化するのは困難かつデリケートな作業である。また性能の良い並列化プログラムを開発するには、必然的に必要となる通信を如何に効率よく行うかが課題となる。それらはマシンの特性に大きく依存するので、詳細は第5章と第6章でそれぞれ述べることにする。

最後に今回並列化される3ルーチン以外について、若干のコメントを加える。それらのルーチンは次のように分類される。

1. 外部ループ外にあるルーチン
2. 外部ループ内にある原理的に並列化が可能なルーチン
3. 外部ループ内にある原理的に並列化が不可能なルーチン

1.にはファイル入出力ルーチンや波動関数および有効ポテンシャルの初期状態を作成するルーチンがある。これらのルーチンは1回しか呼ばれないで、余程計算時間を消費しない限り無理に並列化する必要はない。外部反復を1回に制限したテスト用データで、この部分の消費時間は全計算時間の20%以下だったので、本研究では並列化しなかった。

2.についてはまだ議論の余地がある。たとえば charpw という平面波における電荷計算ルーチンがあるが、charsw とはほぼ同様な手法で並列化が可能なはずである。しかし charpw の計算時間は 14sec と charpw の 1/10 以下であり、今回は並列化しなかった。

本プログラムは全体で約1万6千行あり、そのうち上記3ルーチンは合計で約700行である。つまり全体の5%に満たない部分が計算時間の80%近くを消費している。従ってコストパフォーマンスの面からみて、まず3ルーチンの並列化に集中すべきであると判断した。

5 SX-4 での並列化

5.1 SX-4 の特徴

SX-4 には次のような特徴がある [3]。

- 当センターの SX-4 は 3 ノード構成となっている。1 ノードは 2 個の CPU と 512MB のメモリおよび 1GB の XMU(後述) からなる。
- 1 ノード内の並列化は共有メモリ型であり、ノード間の並列化は分散メモリ型である。
- ノード内は並列化指示行を用いるマイクロタスクもしくは SX 固有の並列化システムルーチンを用いるマクロタスクで並列化される。一方、ノード間にまたがる並列化には MPI を使用する。本研究では比較的並列化が容易で性能のでやすいマイクロタスクによる並列化を試みる。
- マイクロタスクにおいて並列実行するのは、実際に並列化 DO ループ等を含むルーチン単位。つまり、`charsw forcex orthon` は並列実行(冗長実行部も含む)するが、それ以外のルーチン(メインルーチン等)は逐次実行(1CPU のみによる実行)する。(注意 冗長実行ではない!!)
- 変数は static 領域にとられる変数(common 文や save 文に現れる変数 並列化ルーチンの仮引数の変数)と stack 領域にとられる変数(上記以外)に大別される。static 領域変数は並列プロセス数に関係なく 1 個しかないが、stack 領域変数は並列化プロセス数だけあり、プロセスごとに異なる値をとる。
- 並列実行部分でのファイル I/O は順番が保証されない。

5.2 並列化方法

上記の特徴から以下のような作業を行った。

- SX-4 には XMU(eXtended Memory Unit) と呼ばれる高速半導体ディスクが存在する。そのためプログラムを XMU 上で実行するようにした。その結果 CPU TIME にはあまり変化がなかったが、経過時間(elapse time)が 812sec から 525sec と、287sec の高速化が図られた。
- オリジナルのプログラムでは作業領域を節約するため、作業用の変数を common でとっていた。しかしそれでは作業変数が static 領域にとられてしまい、各 CPU が 1 つしかない作業変数を勝手に書き換えてしまう。そのため作業用変数はすべて dimension 文で書き換え、stack 領域にとるようにした。
- `charsw` で全エネルギーを計算するために、各プロセッサで得られたエネルギーの総和計算(GLOBAL SUM)を計算する必要がある。そこで static 領域に新たな変数を確保し、各プロセスで得られたエネルギー値(stack 変数)を static 変数に足し込むように変更した。
- オリジナルの `charsw` の並列化ループ内にファイル出力用ルーチン `top5pr` があった。従ってそのまま並列化すると出力順序が保証されず、結果ファイルが不正となる。そこで static 領域に出力用の配列を確保し、一旦そこに格納した後、1CPU から出力するように変更した。

```

(新 charsw)

23 cGENKEN Modify
24 c      real*8 cmp(0:lup,ntype)
25      common/gkcom3/cmp(0:lup,ntype,nemax,nkmax,nsp) ← ファイル出力用
26 cGENKEN Modify                                     static 変数
                                                 (中略)

43 cGENKEN Modify
44 c      common/cwork1/ylmk(nbase,lmmax),exkr(nbase,nsite),
45 c      &          p(lmmax,nsite),q(lmmax,nsite),
46 c      &          t(nrad,lmmax,nsite),u(nrad,lmmax,nsite),
47 c      &          aln(nbase,0:lup,ntype,nsp),
48 c      &          bln(nbase,0:lup,ntype,nsp),
49 c      &          sjlnr(nbase,nrad,0:lup,mstype)
50      dimension ylmk(nbase,lmmax),exkr(nbase,nsite), ← stack 領域
51      &          p(lmmax,nsite),q(lmmax,nsite),
52      &          t(nrad,lmmax,nsite),u(nrad,lmmax,nsite),
53      &          aln(nbase,0:lup,ntype,nsp),
54      &          bln(nbase,0:lup,ntype,nsp),
55      &          sjlnr(nbase,nrad,0:lup,mstype)
56 c
57      common/gkcom1/elcsum(0:lup,ntype,nsp) ← 総和計算用 static 変数
58 cGENKEN Modify
                                                 (中略)

75 *pdir serial                                ← 逐次実行するための指示行
76      if(lprint.gt.0) then
77          if(mod(loop,lprint).eq.0) then
78              write(6,'('' ===== charsw''
79          &          '' top 5 components of wavefunctions'',
80          &          ''      k sp band      eps''))'
81      endif
82  endif
83 *pdir endserial
84 cGENKEN Modify
85 *pdir pardo                                ← 並列実行するための指示行
86      do 20 k=1,nkpt
87          call mkylmk(k,ylmk)
                                                 (中略)

133 c      if(lprint.gt.0) then
134 c          if(mod(loop,lprint).eq.0) then
135 c              call top5pr(is,k,ie,eps(ie,k,is),cmp) ← ここで出力
136 c          endif
137 c          endif
138      endif
139      30 continue
140      20 continue

```

```

141 c
142 cGENKEN Modify
143 *pdir critical           ← 排他的に実行するための指示行
144     do kk=1,nsp            各 stack の elc を static 変数 elcsum に足し
145         do jj=1,ntype        総和をとる 排他的に行わないと結果不正
146             do ii=0,lup       になる
147                 elcsum(ii,jj,kk) = elcsum(ii,jj,kk) + elc(ii,jj,kk)
148             enddo
149         enddo
150     enddo
151 *pdir endcritical
152 *pdir serial              ← 逐次実行するための指示行 ファイル出力は 1PE から
153     if(lprint.gt.0 .and. mod(loop,lprint).eq.0) then      のみ行う
154         do 21 k=1,nkpt
155             do 31 is=1,nspin
156                 do 32 ie=1,ne
157                     if(abs(wei(ie,k,is)).gt.1.d-12) then
158                         call top5pr(is,k,ie,eps(ie,k,is),
159                         &                               cmp(0,1,ie,k,is))
160                     endif
161     32         continue
162     31         continue
163     21         continue
164     endif
                           (中略)

187 cGENKEN Modify
188 c           call elecpr(nspin,elc)
189     call elecpr(nspin,elcsum)           ← 総和値は elcsum に入っている
190     call chaspr(nspin,'ror      ',ror)
191     endif
192     endif
193 *pdir endserial           ← 逐次実行の終了を指示する指示行

```

(*pdir critical をいれないと、elcsum が正しい値にならない)

- forcex では各プロセス毎の変数の総和計算が必要なので、static 変数を新たに確保し、最後にプロセス間で総和をとるようにした。

(新 forcex)

```

83     common/gkcom2/fmsum(3,nsite),fm0sum(3,nsite) ← 総和計算を行う
                          ための static 変数
                           (中略)

111 cGENKEN Modify
112     do ita=1,nsite           ← static 変数の初期化
113         fmsum (1,ita)=0.d0
114         fmsum (2,ita)=0.d0
115         fmsum (3,ita)=0.d0

```

```

116      fm0sum(1,ita)=0.d0
117      fm0sum(2,ita)=0.d0
118      fm0sum(3,ita)=0.d0
119      enddo
120  *pdir pardo
121  cGENKEN Modify
122  c
123      do 20 k=1,nkpt           ← 並列化ループ
                           (中略)
266  20 continue
277  cGENKEN Modify
278  *pdir critical           ← 各プロセスの stack 変数を static 変数に排他的に
279      do ita=1,nats          たしこむ
280          fmsum (1,ita) = fmsum (1,ita) + fm (1,ita)
281          fmsum (2,ita) = fmsum (2,ita) + fm (2,ita)
282          fmsum (3,ita) = fmsum (3,ita) + fm (3,ita)
283          fm0sum(1,ita) = fm0sum(1,ita) + fm0(1,ita)
284          fm0sum(2,ita) = fm0sum(2,ita) + fm0(2,ita)
285          fm0sum(3,ita) = fm0sum(3,ita) + fm0(3,ita)
286      enddo
287  *pdir end critical
288  *pdir serial            ← このルーチンの最後まで逐次実行
289      do ita=1,nats
290          fm (1,ita) = fmsum (1,ita)  ← 総和値を stack 変数にコピー
291          fm (2,ita) = fmsum (2,ita)

```

5.3 性能測定結果

当センターの SX-4 で性能測定を行った。結果は表 2 の通りであった。なお各ルーチンの elapse time 計測には etime() システム関数を用い、占有状態で計測した。

	オリジナルの cpu time	2CPU 並列時の elapse time	加速率
TOTAL	524.7 sec	327.8 sec	1.60
charsw	173.3 sec	87.8 sec	1.97
forcex	166.0 sec	86.4 sec	1.92
orthon	74.5 sec	38.1 sec	1.96

表 2: 時間コストおよび並列化効果

5.4 考察

計測結果をみると、各ルーチンは 2CPU 並列時で 1.9 倍以上の加速率が得られている。また全体でも 1.60 倍の加速率が得られている。第 5 章で述べたように、本プログラムの 2 並列の場合の理想加速率は 1.65 倍なので、この数値は満足すべきものといえる。

本来ならばより多くの CPU を用いて並列化効果を系統的に調査すべきであろう。しかし当センターの SX-4 の構成はメモリを共有できる CPU 数は 1 ノード内の 2CPU のみである。他ノードの CPU を使用するには MPI でプログラムを全面的に書き換える必要があり、その工数はマイクロタスクによる並列化の比ではない。これらの理由によりこれ以上の並列化は、ここではあきらめざるをえない。

最後に自動並列化について一言記しておく。共有メモリ型並列化の大きなメリットの一つに自動並列化が比較的容易にサポートできることがある。SX-4においてもマイクロタスクによる自動並列化がサポートされている。これは fopp(Fortran Optimizing PreProcessor) と呼ばれるプリプロセッサが do ループレベルの並列性を抽出し、マイクロタスクによる並列化プログラムを生成するものである。この機能によるとユーザは並列化の概念や指示行についての知識無しで、容易に自分のプログラムを並列化できる。従ってユーザの負担を軽減すると言う意味で非常に有力である。ただし自動並列化できるのは単純な do ループに限られ、ループ内でサブルーチンをコールしている場合は自動並列化できない。本研究で並列化されるループ内には多数のサブルーチンコールがあったため、自動並列化は当初から断念せざるを得なかつた。

6 VPP300 での並列化

6.1 VPP300 の特徴

VPP300 には次のような特徴がある [4]。

- 分散メモリ方式であり、各 PE(Processing Element) は 1 個の CPU と 512MB のメモリからなる。
- 並列プログラムは逐次実行部分 (1PE で実行)、冗長実行部分 (複数の PE で同一の内容を実行)、並列実行部分 (複数の PE で異なる内容を実行) に大別される。
- parallel region で囲まれた部分は冗長実行部分もしくは並列実行部分であり、それ以前の逐次実行部分の変数は parallel region の最初で各 PE にコピーされる。
- ファイル出力は冗長実行部では 1PE の出力のみであり、並列実行部では全 PE から出力されるが順序は保証されない。
- ファイル入力は冗長実行部では逐次プログラムと同様に動作し全 PE に反映される。一方並列実行部分では各々の PE への入力が有効だが、入力順序を保証するにはコーディングを工夫する必要がある。

6.2 UNIFY 手法

分散メモリ型の並列化の場合、各 PE にどのように変数を割り当てるかが問題となる。データを分割して各 PE に受け持たせる方法は非常に大きなメモリを利用できる利点があるが、その反面プログラム全体を把握しつつ大幅に書き換える必要があり、非常に困難である。一方 VPP300 は 1PE に 512MB という比較的大きなメモリを有しており、通常の場合、全データを 1PE 内に保有できる。そこで今回はデータ分割はせず、並列計算のみを行う並列化手法 (UNIFY 手法)[5] を採用した。これは並列実行直後に全 PE の配列の中身を各 PE 間の総和計算 (GLOBAL SUM) により同一にし、その後は全 PE で同一の計算を冗長に実行するという手法である。この手法のメリットは並列化によるプログラムの変更を局所化できることである。いいかえれば並列化する部分以外のプログラムをみる必要がない。

UNIFY 手法を説明するために、次のような簡単な DO ループを考える。

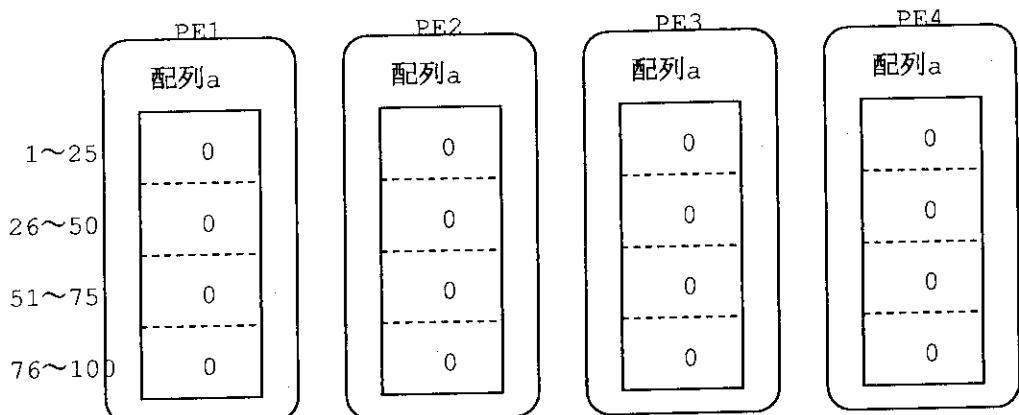
```
do i = 1,100
    a(i) = b(i)*c(i)
enddo
```

これを VPP300 の指示行を用いて UNIFY 手法により並列化すると次のようになる。

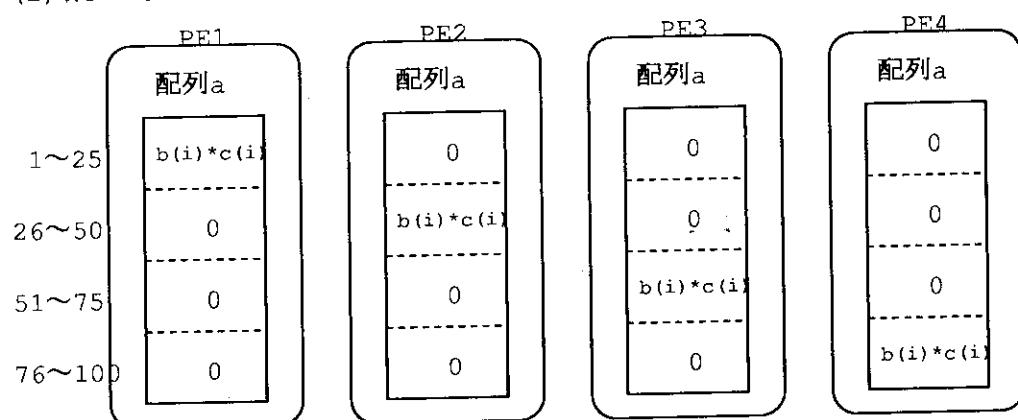
```
do i = 1,100           ← 全 PE の配列 a を 0 クリア ... (1)
    a(i) = 0.0
enddo
!xocl xocl spread do   ← 並列実行のための指示行 ... (2)
    do i = 1,100
        a(i) = b(i)*c(i)
    enddo
!xocl end spread sum(a)   ← 各 PE の a(i) の総和を計算し全 PE の a(i) に反映させる
                            ... (3)
```

4PE の場合について図示すると次のようになる。

(1) 全PEの配列aを0クリア



(2) 各PEが受け持ち部分の計算を行ない その結果を配列aに格納する



(3) 全PEの総和計算を行ない 配列aの内容を同一にする

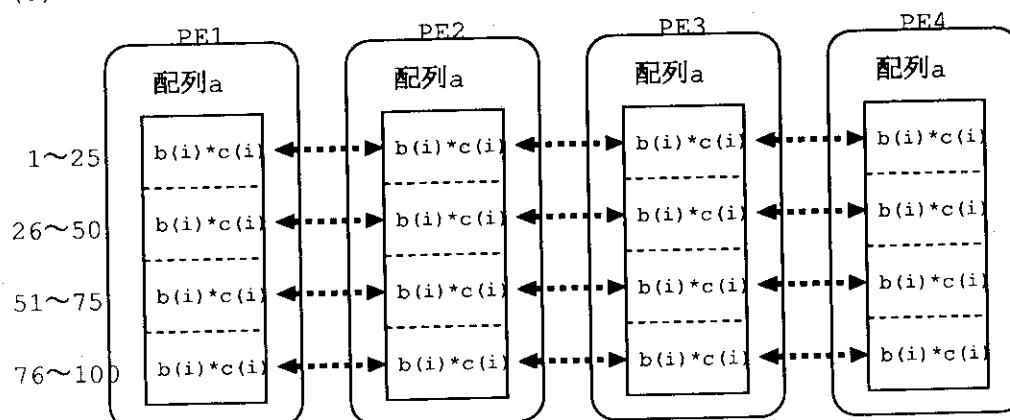


図 3: UNIFY 手法の実現方式

6.3 並列化方法

上記の特徴から、以下のような作業を行った。

- 高コスト 3ルーチンの do 20 k=1,nkpt に対し UNIFY 手法による並列化を行った。
- charsw では do 20 k=1,nkpt の内部に配列 cmp の出力ルーチン top5pr があった。従ってそのまま並列化したのでは出力順序が保証されない。そこで配列 cmp を大きくした上でループ内では出力せずに配列に格納するだけとし、ループ外の冗長実行部分で出力するようにした。

(新 charsw)

```

22 cGENKEN Modify
23 c      real*8 cmp(0:lup,ntype)
24      real*8 cmp(0:lup,ntype,nemax,nkmax,nsp)
           (中略)
47      common/cprint/lprint,ngpr,incr,incr0
48 cGENKEN Modify
49 !xocl subprocessor sp(:)
           (中略)
70 cGENKEN Modify      ← 大きくした配列をループ外で 0 クリア
71      do mmm = 1,nsp
72          do lll = 1,nkmax
73              do kkk = 1,nemax
74                  do jjj = 1,ntype
75                      do iii = 0,lup
76                          cmp(iii,jjj,kkk,lll,mmm) = 0.d0
77                      enddo
78                  enddo
79              enddo
80          enddo
81      enddo
82 !xocl spread do      ← 並列実行
83      do 20 k=1,nkpt
84          call mkylmk(k,ylmk)
85          call mkexkr(k,exkr)
           (中略)
124 cGENKEN Modify
125 c      cmp(l,it)=cmp(l,it)/vol/natom(it)
126 c      elc(l,it,is)=elc(l,it,is)+(2.d0/nspin)*wei(ie,k,is)
127 c      &                               *cmp(l,it)
128      cmp(l,it,ie,k,is)=cmp(l,it,ie,k,is)/vol/natom(it)
129      elc(l,it,is)=elc(l,it,is)+(2.d0/nspin)*wei(ie,k,is)
130      &                               *cmp(l,it,ie,k,is)
131      52      continue
132 cGENKEN Modify      ← 並列ループ内の出力は順序が不定
133 c      if(lprint.gt.0) then
134 c          if(mod(loop,lprint).eq.0) then

```

```

135 c           call top5pr(is,k,ie,eps(ie,k,is),cmp)
136 c           endif
137 c           endif
138     endif
139   30 continue
140   20 continue
141 !xocl end spread sum(elc),sum(ror),sum(cmp) ←各 PE で計算した値の総和
142 c
143 cGENKEN Modify      ←ループ外の冗長実行部分で cmp をファイル出力
144   if(lprint.gt.0 .and. mod(loop,lprint).eq.0) then
145     do 21 k=1,nkpt
146       do 31 is=1,nspin
147         do 32 ie=1,ne
148           if(abs(wei(ie,k,is)).gt.1.d-12) then
149             call top5pr(is,k,ie,eps(ie,k,is),
150               &           cmp(0,1,ie,k,is))
151           endif
152   32 continue
153   31 continue
154   21 continue
155   endif

```

- orthon における do 20 k=1,nkpt では配列 cnn に対し schmidt の正規直交化を行っており、cnn のアップデート計算をしている。従って cnn を 0 クリアするわけにはいかない。そこで cnn を各 PE で計算した後ワーク領域 cnntmp にコピーし、cnntmp に対して総和計算を行い、その結果を cnn に戻すこととした。
- 上記以外のファイル出力は冗長実行部分にあったので、なんの変更も必要なかった。

(新 orthon)

```

43 cGENKEN Modify
44 !xocl subprocessor sp(:)
45   complex*16 cnntmp(nbbase,nkmax) ←総和計算用ワーク領域
46 c
47   if(dt(loop).le.0.d0) return
48 c
49   tpi=8.d0*atan(1.d0)
50   fpi=tpi+tpi
51   vol=abs(deter3(brmt))
52   vi=1.d0/vol
53 c
54 cGENKEN Modify
55 !xocl spread do           ←並列計算
56   do 20 k=1,nkpt
57     call mkylmk(k,ylmk)
58     call mkexkr(k,exkr)

```

```

59      call mkabln(nspin,k,aln,bln)
60      call mkjln0(k,sjlnr)
          (中略)

132     20 continue
133 cGENKEN Modify
134 !xocl end spread
135     do ll=1,nsp
136         do jj=1,nemax
137             do k=1,nkpt
138                 do ii=1,nbase
139                     cnntmp(ii,k) = 0.d0    ←ワーク領域の0クリア
140                 enddo
141             enddo
142 !xocl spread do           ←計算した部分のcnnをcnntmpにコピー
143             do k = 1,nkpt
144                 do ii=1,nbase
145                     cnntmp(ii,k) = cnn(ii,jj,k,ll)
146                 enddo
147             enddo
148 !xocl end spread sum(cnntmp)   ←cnntmpの総和計算
149             do k = 1,nkpt    ←cnntmpをcnnに戻す
150                 do ii=1,nbase
151                     cnn(ii,jj,k,ll) = cnntmp(ii,k)
152                 enddo
153             enddo
154         enddo
155     enddo
156 c
157     return
158 end

```

6.4 性能測定結果

当センターのVPP300で性能測定を行った。結果は表3及び図4の通りであった。なお各ルーチンのelapsed time 計測にはgettod() システム関数を用い、占有状態で計測した。本プログラムの並列化対象部分は78.7%であり、(6)式に従って理想加速率を算出している。

	1PE	4PE	8PE	12PE
TOTAL	512.3(1.00)	244.4(2.10)	200.8(2.55)	183.1(2.80)
charsw	164.4(1.00)	42.7(3.85)	24.1(6.82)	15.0(10.96)
forcex	191.3(1.00)	45.4(4.21)	26.9(7.11)	16.0(11.96)
orthon	47.5(1.00)	12.4(3.83)	7.1(6.69)	4.4(10.80)

表 3: 時間コストおよび並列化効果 (テスト用データ) sec(ratio)

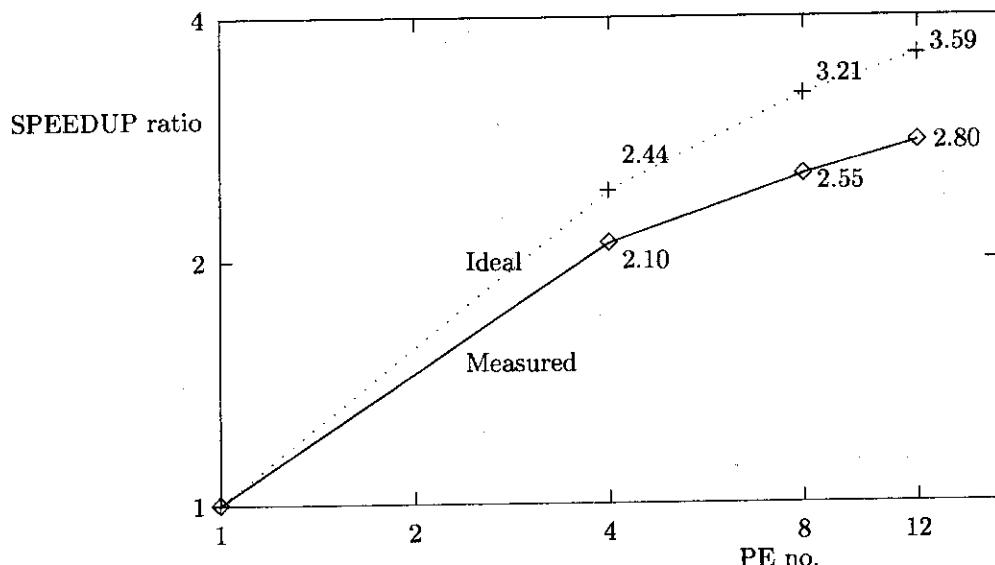


図 4: PE 数と加速率の関係 (理想値と実測値)

今までの測定結果はテスト用入力データによるものであったが、実用入力データを用い外部反復を 50 回にして計測を行った。その結果を以下に記す (表 4 及び図 5 参照)。このデータにおける並列化率は 87.1 % である。

	1PE	4PE	8PE	12PE
TOTAL	23291(1.00)	7943(2.93)	5824(4.00)	4686(4.97)
charsw	8259(1.00)	2107(3.92)	1190(6.94)	721(11.45)
forcex	9598(1.00)	2271(4.22)	1347(7.13)	800(12.00)
orthon	2438(1.00)	622(3.92)	356(6.85)	222(10.98)

表 4: 時間コストおよび並列化効果(実用データ) sec(ratio)

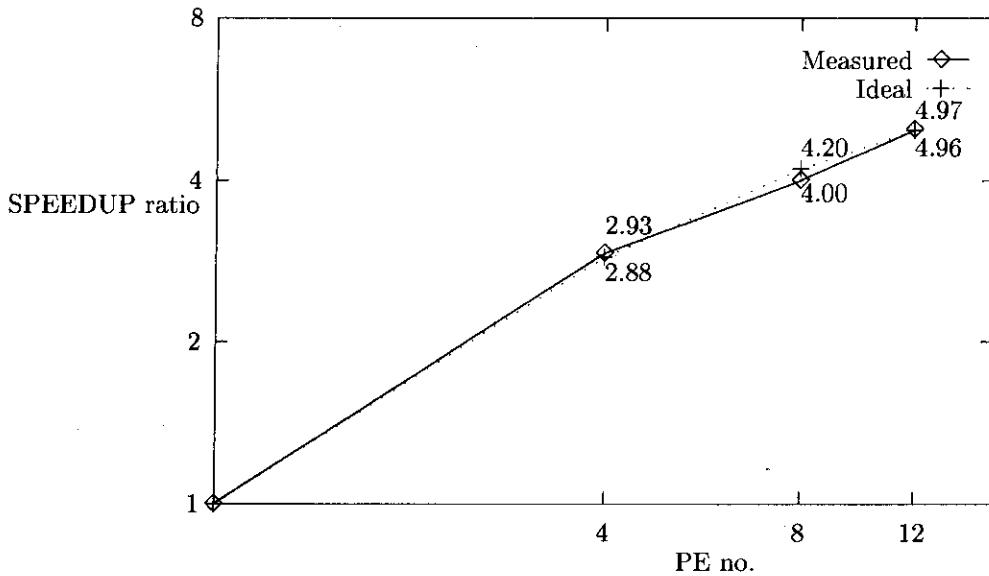


図 5: PE 数と加速率の関係(理想値と実測値)

6.5 考察

測定結果をみると実用データの場合は、理想的な加速率が得られているといえる。本プログラムが原理的に並列性が高く、通信量が最小限に抑えられているためと考えられる。特に forcex は通信が極めて少ないので、他の 2 ルーチンよりも良い性能が得られている。

一方テスト用データにおいては、測定値が理想値にはほど遠い。最初に考えられる原因是通信コストである。VPP には MPA(Moving Performance Analyzer) と呼ばれる通信コスト調査ツールがサポートされており、容易に並列プログラム実行時に消費された通信時間を測定することができる。これを用いて通信時間を測定したところ、ほとんど無視できる程度の時間であることが分かった。たとえば 4PE で通信パケットを送っている時間は、わずか 0.43 秒であった。従って上記性能低下は他に原因がある。

そこで全ルーチンの消費時間を測定し直した。その結果、writ18 というバイナリデータ出力ルーチンにおいて、非並列プログラムでは約 25 秒であったのが並列プログラムでは約 50 秒と、経過時間が倍増していることが分かった。幸いこのルーチンは外部ループ外にあったので、この部分をパラレルリージョン外に移して、再度測定しなおした。(表 5 及び図 6 参照)。

	1PE	4PE	8PE	12PE
TOTAL	512.3(1.00)	223.5(2.29)	178.3(2.87)	159.5(3.21)
writ18	24.9(1.00)	28.8(0.86)	27.6(0.90)	27.4(0.91)

表 5: 時間コストおよび並列化効果 (改良版) sec(ratio)

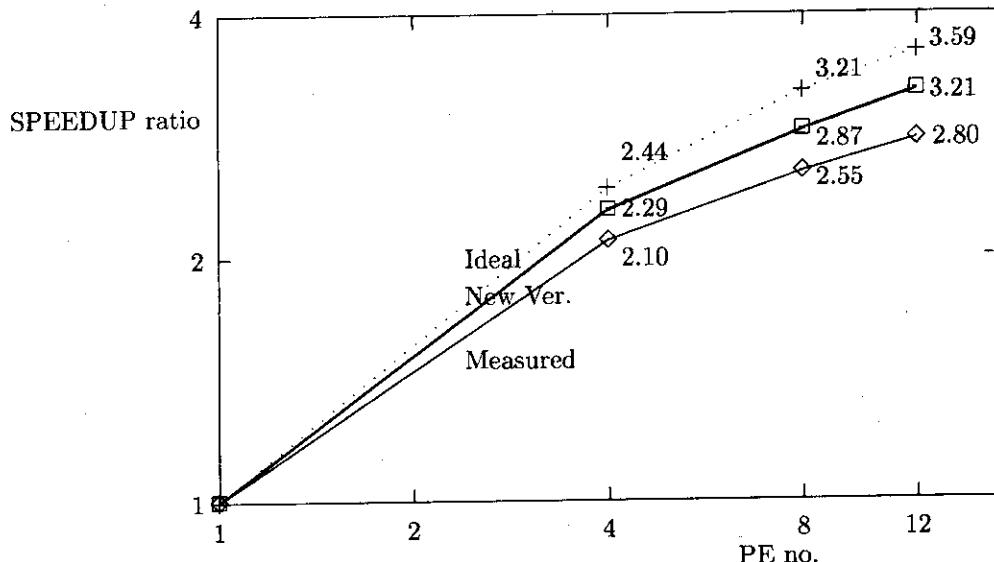


図 6: PE 数と加速率の関係 (理想値と実測値)

上記の図およびグラフから、新ルーチンの加速率がかなり改善されてことがわかる。実は並列化作業の初期の段階で、バイナリデータ入力ルーチンに関して同様の問題が生じていた。その時はそのルーチンに非常に時間がかかっていたので(非並列の場合の数十倍!!)問題点がすぐに明らかになり、早急に対策をとることができた。一方 writ18に関しては、目立たない程度のオーバーヘッドだったため発見が遅れたといえる。

このように、パラレルリージョン内のファイル入出力は十分に注意すべきであることが分かる。理想的にはパラレルリージョン内では計算のみに専念し、ファイル入出力は行わないことが望ましい。

7まとめ

今回の並列化作業において SX-4 では現環境の制約で 2 並列でのみの実行となった。2 並列での加速率は 1.60 倍とほぼ妥当な値が得られたが、より CPU 数が多い場合どのような性能を示すかは未知である。いずれ機会をみて系統的な性能測定を試みたい。

一方、VPP300 では 12PE 使用で最大 4.97 倍の加速率が得られた。これは並列化率を考えると理想的な値といえる。また分散メモリ型の並列化に有力と思われる UNIFY 手法について説明した。本手法は並列化における変更を局所化できるので、プログラム全体をみる必要がなく、プログラマの負担は大幅に軽減する。反面、全 PE が常に全部の配列を保持する必要があるため、分散メモリ型並列マシンの大きな長所である PE 数に比例したメモリ容量が有効に利用できない。従って今後の課題として、配列を各 PE に分割することにより、大規模計算を可能とする並列化コードの開発があげられる。

本報告では第一原理電子状態計算プログラムの共有メモリ型及び分散メモリ型マシン用の並列化について、やや細かいと思われることまで記述した。それは第 1 章でも述べたように、従来はあまり語られてこなかった”泥臭い”ノウハウの蓄積と共有が、並列計算の普及に不可欠と信じるからである。本報告が並列計算を志す研究者やプログラマに少しでも役立つことを祈念してやまない。

謝　　辞

日本原子力研究所 計算科学技術推進センター 並列処理支援技術開発グループ 折居茂夫氏には VPP300 の並列化手法やツールの利用方法等の多岐にわたり、多くの助言を頂きました。この場を借りて感謝致します。

参考文献

- [1] T. OGUCHI and T. SASAKI, Density-Functional Molecular-Dynamics Method Progress of Theoretical Physics Supplement No. 103 ,93 (1991).
- [2] H. Krakauer, M. Posternak and A.J. Freeman, Phys. Rev. B 19 ,1706 (1979).
- [3] Super-UX FORTRAN90/SX プログラミングの手引き
- [4] UXP/V Fortran90/VPP 使用手引書
- [5] 折居茂夫, 大田敏郎 分子動力学コードの段階的並列化手法 JAERI-Data/Code 96-023

7まとめ

今回の並列化作業において SX-4 では現環境の制約で 2 並列でのみの実行となった。2 並列での加速率は 1.60 倍とほぼ妥当な値が得られたが、より CPU 数が多い場合どのような性能を示すかは未知である。いずれ機会をみて系統的な性能測定を試みたい。

一方、VPP300 では 12PE 使用で最大 4.97 倍の加速率が得られた。これは並列化率を考えると理想的な値といえる。また分散メモリ型の並列化に有力と思われる UNIFY 手法について説明した。本手法は並列化における変更を局所化できるので、プログラム全体をみる必要がなく、プログラマの負担は大幅に軽減する。反面、全 PE が常に全部の配列を保持する必要があるため、分散メモリ型並列マシンの大きな長所である PE 数に比例したメモリ容量が有効に利用できない。従って今後の課題として、配列を各 PE に分割することにより、大規模計算を可能とする並列化コードの開発があげられる。

本報告では第一原理電子状態計算プログラムの共有メモリ型及び分散メモリ型マシン用の並列化について、やや細かいと思われることまで記述した。それは第 1 章でも述べたように、従来はあまり語られてこなかった”泥臭い”ノウハウの蓄積と共有が、並列計算の普及に不可欠と信じるからである。本報告が並列計算を志す研究者やプログラマに少しでも役立つことを祈念してやまない。

謝　　辞

日本原子力研究所 計算科学技術推進センター 並列処理支援技術開発グループ 折居茂夫氏には VPP300 の並列化手法やツールの利用方法等の多岐にわたり、多くの助言を頂きました。この場を借りて感謝致します。

参考文献

- [1] T. OGUCHI and T. SASAKI, Density-Functional Molecular-Dynamics Method Progress of Theoretical Physics Supplement No. 103 ,93 (1991).
- [2] H. Krakauer, M. Posternak and A.J. Freeman, Phys. Rev. B 19 ,1706 (1979).
- [3] Super-UX FORTRAN90/SX プログラミングの手引き
- [4] UXP/V Fortran90/VPP 使用手引書
- [5] 折居茂夫, 大田敏郎 分子動力学コードの段階的並列化手法 JAERI-Data/Code 96-023

7まとめ

今回の並列化作業において SX-4 では現環境の制約で 2 並列でのみの実行となった。2 並列での加速率は 1.60 倍とほぼ妥当な値が得られたが、より CPU 数が多い場合どのような性能を示すかは未知である。いずれ機会をみて系統的な性能測定を試みたい。

一方、VPP300 では 12PE 使用で最大 4.97 倍の加速率が得られた。これは並列化率を考えると理想的な値といえる。また分散メモリ型の並列化に有力と思われる UNIFY 手法について説明した。本手法は並列化における変更を局所化できるので、プログラム全体をみる必要がなく、プログラマの負担は大幅に軽減する。反面、全 PE が常に全部の配列を保持する必要があるため、分散メモリ型並列マシンの大きな長所である PE 数に比例したメモリ容量が有効に利用できない。従って今後の課題として、配列を各 PE に分割することにより、大規模計算を可能とする並列化コードの開発があげられる。

本報告では第一原理電子状態計算プログラムの共有メモリ型及び分散メモリ型マシン用の並列化について、やや細かいと思われることまで記述した。それは第 1 章でも述べたように、従来はあまり語られてこなかった”泥臭い”ノウハウの蓄積と共有が、並列計算の普及に不可欠と信じるからである。本報告が並列計算を志す研究者やプログラマに少しでも役立つことを祈念してやまない。

謝　　辞

日本原子力研究所 計算科学技術推進センター 並列処理支援技術開発グループ 折居茂夫氏には VPP300 の並列化手法やツールの利用方法等の多岐にわたり、多くの助言を頂きました。この場を借りて感謝致します。

参考文献

- [1] T. OGUCHI and T. SASAKI, Density-Functional Molecular-Dynamics Method Progress of Theoretical Physics Supplement No. 103 ,93 (1991).
- [2] H. Krakauer, M. Posternak and A.J. Freeman, Phys. Rev. B 19 ,1706 (1979).
- [3] Super-UX FORTRAN90/SX プログラミングの手引き
- [4] UXP/V Fortran90/VPP 使用手引書
- [5] 折居茂夫, 大田敏郎 分子動力学コードの段階的並列化手法 JAERI-Data/Code 96-023

付録 オリジナルソース

```

(charsw)
1 c---- charsw
2 c construction of charge density with spherical wave form
3 c      ror : spherical part of charge density from psi and -psi
4 c
5      subroutine charsw(nspin, eps, wei, cnn, ror, loop)
6      implicit real*8 (a-h, o-z)
7      parameter(nbase=1499)
8      parameter(nemax=50, nkmax=36)
9      parameter(ntype=4, nsite=7, mstype=2, nrad=401, nrad0=61)
10     parameter(nsp=1)
11     parameter(lup=2, lmax=(lup+1)**2)
12     parameter(c=274.074d0)
13 c
14     complex*16 cnn(nbase, nemax, nkmax, nsp)
15     real*8 eps(nemax, nkmax, nsp)
16     real*8 wei(nemax, nkmax, nsp)
17 c
18     real*8 ror(nrad, ntype, nsp)
19 c
20     complex*16 ylmk, exkr, p, q, t, u
21     complex*16 phi1, phi2
22 c
23     real*8 cmp(0:lup, ntype)
24     real*8 elc(0:lup, ntype, nsp)
25 c
26     common/cbasis/brmt(3, 3)
27     common/ckpvec/ak(3, nkmax), wk(nkmax), nkpt
28     common/cpwind/indpw(3, nbase, nkmax)
29     common/csizes/pow(nkmax), ne
30     common/catoms/zval(ntype), idmsh(ntype), natom(ntype), lmax, ntp
31     common/cradii/radmt(mstype), xstep(mstype),
32     &          nrd(mstype), nrd0(mstype), mstp
33     common/crdfun/pfun(nrad, 0:lup, ntype, nsp),
34     &          qfun(nrad, 0:lup, ntype, nsp)
35     common/crefun/pefun(nrad, 0:lup, ntype, nsp),
36     &          qefun(nrad, 0:lup, ntype, nsp)
37     common/casfun/pf(0:lup, ntype, nsp), dpf(0:lup, ntype, nsp),
38     &          pef(0:lup, ntype, nsp), dpef(0:lup, ntype, nsp),
39     &          penorm(0:lup, ntype, nsp), wronsk(0:lup, ntype, nsp)
40     common/cwork1/ylmk(nbase, lmax), exkr(nbase, nsite),
41     &          p(lmax, nsite), q(lmax, nsite),
42     &          t(nrad, lmax, nsite), u(nrad, lmax, nsite),
43     &          aln(nbase, 0:lup, ntype, nsp),
44     &          bln(nbase, 0:lup, ntype, nsp),
45     &          sjlnr(nbase, nrad, 0:lup, mstype)
46     common/cprint/lprint, ngpr, incr, incr0
47 c
48     vol=abs(deter3(brmt))
49     fpi=16, d0*atan(1, d0)
50     lmax=(lmax+1)**2
51     do 10 is=1, nsp
52     do 10 it=1, ntype
53       do 12 ir=1, nrad
54         ror(ir, it, is)=0. d0
55   12 continue
56   do 14 l=0, lup
57     elc(l, it, is)=0. d0
58   14 continue
59   10 continue
60   if(lprint.gt.0) then
61     if(mod(loop, lprint).eq.0) then
62       write(6, ('' ===== charsw'',
63     &           '' top 5 components of wavefunctions'')

```

```

64      &          /*    k sp band      eps'') )
65      endif
66      endif
67      do 20 k=1,nkpt
68      call mkylmk(k,ylmk)
69      call mkexkr(k,exkr)
70      call mkabin(nspin,k,aln,bln)
71      call mkljnl1(k,sjlnr)
72      do 30 is=1,nspin
73      do 30 ie=1,ne
74      if(abs(wei(ie,k,is)).gt.1.d-12) then
75      call mkpqt1(is,k,ie,cnn,ylmk,exkr,aln,bln,sjlnr,p,q,t)
76      do 50 it=1,nttp
77      do 50 l=0,lmax
78      cmp(l,it)=0.d0
79      50 continue
80      ita=0
81      do 60 it=1,nttp
82      do 60 ia=1,natom(it)
83      ita=ita+1
84      im=idmsh(it)
85      lm=0
86      do 62 l=0,lmax
87      do 62 m=-l,l
88      lm=lm+1
89      do 64 ir=1,nrd(im)
90      phi1= p(lm,ita)*pfun(ir,l,it,is)
91      +q(lm,ita)*pefun(ir,l,it,is)
92      phi2= p(lm,ita)*qfun(ir,l,it,is)
93      +q(lm,ita)*qefun(ir,l,it,is)
94      ror(ir,it,is)=ror(ir,it,is)
95      +wei(ie,k,is)*((abs(phi1)**2+abs(phi2)**2)
96      -abs(t(ir,lm,ita))**2)
97      64 continue
98      cmp(l,it)=cmp(l,it)+abs(p(lm,ita))**2
99      +abs(q(lm,ita))**2*penorm(l,it,is)
100     62 continue
101     60 continue
102     do 52 it=1,nttp
103     do 52 l=0,lmax
104     cmp(l,it)=cmp(l,it)/vol/natom(it)
105     elc(l,it,is)=elc(l,it,is)+(2.d0/nspin)*wei(ie,k,is)
106     *cmp(l,it)
107     52 continue
108     if(lprint.gt.0) then
109       if(mod(loop,lprint).eq.0) then
110         call top5pr(is,k,ie,eps(ie,k,is),cmp)
111       endif
112     endif
113   endif
114   30 continue
115   20 continue
116 c
117   f=(2.d0/nspin)/vol/sqrt(fpi)
118   do 70 is=1,nspin
119   do 70 it=1,nttp
120   do 72 ir=1,nrad
121   ror(ir,it,is)=ror(ir,it,is)*f/natom(it)
122   72 continue
123   70 continue
124 c make ror as charge and spin density array
125   if(nspin.eq.2) then
126     do 80 it=1,nttp
127     do 82 ir=1,nrad
128     char=ror(ir,it,1)+ror(ir,it,nspin)
129     spin=ror(ir,it,1)-ror(ir,it,nspin)

```

```

130      ror(ir,it,1) =char
131      ror(ir,it,nspin)=spin
132      82 continue
133      80 continue
134      endif
135      c
136      if(lprint.gt.0) then
137          if(mod(loop,lprint).eq.0) then
138              call elecp(nspin,elc)
139              call chaspr(nspin,'ror',ror)
140          endif
141      endif
142      return
143      end

(forcex)
1  c---- forcex
2  c
3  subroutine forcex(nspin,wei,cnn,vcg,vtg,vtr0,rpa,fcn,foa,eps,loop)
4  implicit real*8 (a-h,o-z)
5  parameter(nbase=1499)
6  parameter(nemax=50,nkmax=36)
7  parameter(ntype=4,nsite=7,mstype=2,nrad=401,nrad0=61)
8  parameter(nsp=1)
9  parameter(lup=2,lmmax=(lup+1)**2)
10  parameter(ng1max=7)
11  parameter(ng2max=7)
12  parameter(ng3max=61)
13  parameter(nbaseg=13999)
14  parameter(nstars=3999)
15  complex*16 cnn(nbase,nemax,nkmax,nsp)
16  complex*16 fcn(nbase,nemax,nkmax,nsp)
17  complex*16 vcg(nbaseg)
18  complex*16 vtg(nbaseg,nsp)
19  complex*16 rpa(nbaseg,nsite)
20  real*8 wei(nemax,nkmax,nsp)
21  real*8 eps(nemax,nkmax,nsp)
22  real*8 foa(3,nsite)
23  c
24  complex*16 smm(nemax,nemax)
25  complex*16 hmm(nemax,nemax)
26  complex*16 zmm(nemax,nemax)
27  complex*16 smt(nbase)
28  complex*16 hmt(nbase)
29  complex*16 samt(nbase,nsite),hamt(nbase,nsite)
30  complex*16 samt0(nbase,nsite),hamt0(nbase,nsite)
31  complex*16 ss(nbase)
32  complex*16 ylmk,exkr,p,q,t,u
33  complex*16 aan,ban,wt,wu,sexp,hexp,cmt,cmt0,phagv
34  c
35  real*8 hr((nemax*(nemax+1))/2)
36  real*8 hi((nemax*(nemax+1))/2)
37  real*8 or((nemax*(nemax+1))/2)
38  real*8 oi((nemax*(nemax+1))/2)
39  real*8 zr(nemax,nemax)
40  real*8 zi(nemax,nemax)
41  real*8 work(nemax,5)
42  c
43  real*8 vtr0(nrad0,ntype,nsp)
44  real*8 rcmt(3,3)
45  real*8 fog(3,nsite),fom(3,nsite),fom0(3,nsite)
46  real*8 fm(3,nsite),fm0(3,nsite)
47  c
48  common/cbasis/brmt(3,3)

```

```

49      common/cspace/drot(3,3,48),taurot(3,48),invrot(48),nsym
50      common/cspatr/trmt(3,3,48)
51      common/csptau/itarot(48,nsite)
52      common/cpwgvt/phagy(nbseg),lmngv(3,nbaseg),ngv
53      common/cpwsta/gv2(nstars),
54      &           lsgv(-ng1max:ng1max,-ng2max:ng2max,-ng3max:ng3max),
55      &           lmgv(2,nstars),lgvrot(48,nstars),nst
56      common/ckpvec/ak(3,nkmax),wk(nkmax),nkpt
57      common/cpwind/indpw(3,nbase,nkmax)
58      common/csizes/npw(nkmax),ne
59      common/catoms/zval(ntype),idmsh(ntype),natom(ntype),lmax,ntp
60      common/cradii/radmt(mstype),xstep(mstype),
61      &           nrd(mstype),nrd0(mstype),mstp
62      common/cradwe/rval(nrad,mstype),rwei(nrad,mstype)
63      common/cradw0/rval0(nrad0,mstype),rwei0(nrad0,mstype)
64      common/csites/tau(3,nsite),nats
65      common/casfun/pf(0:lup,ntype,nsp),dpf(0:lup,ntype,nsp),
66      &           pef(0:lup,ntype,nsp),dpef(0:lup,ntype,nsp),
67      &           penorm(0:lup,ntype,nsp),wronsk(0:lup,ntype,nsp)
68      common/catome/ezr(0:lup,ntype,nsp)
69      common/cdiag/ldiag
70      common/cwork1/ylmk(nbase,lmax),exkr(nbase,nsite),
71      &           p(lmax,nsite),q(lmax,nsite),
72      &           t(nrad,lmax,nsite),u(nrad,lmax,nsite),
73      &           aln(nbase,0:lup,ntype,nsp),
74      &           bln(nbase,0:lup,ntype,nsp),
75      &           sjlnr(nbase,nrad,0:lup,mstype)
76      common/cprint/lprint,ngpr,incr,incr0
77  c
78      tpi=8.d0*atan(1.d0)
79      fpi=tpi+tpi
80      sfp=sqrt(fpi)
81      vol=abs(deter3(brmt))
82      vj=1.d0/vol
83      is1
84      call rcpvec(rcomt)
85          call swatch(51,1,'force0 ')
86      call force0(vcg,rpa,fog)
87          call swatch(51,2,'force0 ')
88      do 10 ita=1,nsite
89          fom(1,ita)=0.d0
90          fom(2,ita)=0.d0
91          fom(3,ita)=0.d0
92          fom0(1,ita)=0.d0
93          fom0(2,ita)=0.d0
94          fom0(3,ita)=0.d0
95          fm(1,ita)=0.d0
96          fm(2,ita)=0.d0
97          fm(3,ita)=0.d0
98          fm0(1,ita)=0.d0
99          fm0(2,ita)=0.d0
100         fm0(3,ita)=0.d0
101     10 continue
102  c
103     do 20 k=1,nkpt
104         call swatch(52,1,'mkylmk ')
105         call mkylmk(k,ylmk)
106         call swatch(52,2,'mkylmk ')
107         call swatch(53,1,'mkexkr ')
108         call mkexkr(k,exkr)
109         call swatch(53,2,'mkexkr ')
110         call swatch(54,1,'mkabln ')
111         call mkabln(nspin,k,aln,bln)
112         call swatch(54,2,'mkabln ')
113         call swatch(55,1,'mkjlnr ')
114         call mkjlnr0(k,sjlnr)

```

```

115           call swatch(55,2,'mkjlnr ')
116       do 21 is=1,nspin
117       do 22 ie=1,ne
118         weit=(2.d0/nspin)*wei(ie,k,is)
119         call swatch(56,1,'mat000 ')
120         call mat000(is,k,ie,cnn,vtg,smt,hmt)
121         call swatch(56,2,'mat000 ')
122         call swatch(57,1,'mkpqtu ')
123         call mkpqtu(is,k,ie,cnn,ylmk,exkr,aln,b1n,sjlnr,vtr0,
124           p,q,t,u)
125         call swatch(57,2,'mkpqtu ')
126         call swatch(58,1,'do 30 ')
127       do 30 ita=1,nats
128       do 30 ipw=1,nbase
129         samt(ipw,ita)=0.d0
130         samt0(ipw,ita)=0.d0
131         hamt(ipw,ita)=0.d0
132         hamt0(ipw,ita)=0.d0
133     30   continue
134           call swatch(58,2,'do 30 ')
135           call swatch(59,1,'do 40 ')
136           ita=0
137           do 40 it=1,ntp
138           do 40 ia=1,natom(it)
139             ita=ita+1
140             im=idmsh(it)
141             im=0
142             do 42 l=0,lmax
143             do 42 m=-l,l
144               im=im+1
145               do 44 ipw=1,npw(k)
146                 aan=conjg(ylmk(ipw,im))*exkr(ipw,ita)*aln(ipw,l,it,is)
147                 ban=conjg(ylmk(ipw,im))*exkr(ipw,ita)*bln(ipw,l,it,is)
148                 samt(ipw,ita)=samt(ipw,ita)
149                   +conjg(aan)*p(im,ita)
150                   +conjg(ban)*q(im,ita)*penorm(l,it,is)
151                 hamt(ipw,ita)=hamt(ipw,ita)
152                   +conjg(aan)*p(im,ita)*ezr(l,it,is)
153                   +conjg(aan)*q(im,ita)
154                   +conjg(ban)*q(im,ita)*ezr(l,it,is)
155                   *penorm(l,it,is)
156     44   continue
157     do 46 ipw=1,npw(k)
158       ss(ipw)=ylmk(ipw,im)*conjg(exkr(ipw,ita))
159     46   continue
160     do 48 ir=1,nrd0(im)
161       wt=t(ir,im,ita)*rwei0(ir,im)
162       wu=u(ir,im,ita)*rwei0(ir,im)
163     do 49 ipw=1,npw(k)
164       samt0(ipw,ita)=samt0(ipw,ita)
165         +ss(ipw)*sjlnr(ipw,ir,l,im)*wt
166       hamt0(ipw,ita)=hamt0(ipw,ita)
167         +ss(ipw)*sjlnr(ipw,ir,l,im)*wu
168     49   continue
169     48   continue
170     42   continue
171     40   continue
172           call swatch(59,2,'do 40 ')
173           call swatch(60,1,'do 60 ')
174       do 60 ita=1,nats
175       do 62 ipw=1,npw(k)
176         samt(ipw,ita)=samt(ipw,ita)*fpi*vi
177         hamt(ipw,ita)=hamt(ipw,ita)*fpi*vi
178         samt0(ipw,ita)=samt0(ipw,ita)*fpi*vi
179         hamt0(ipw,ita)=hamt0(ipw,ita)*fpi*vi
180     62   continue

```

```

181    60  continue
182          call swatch(60,2,'do 60   ')
183          call swatch(61,1,'do 70   ')
184      do 70 ita=1,nats
185      do 70 ipw=1,npw(k)
186          smt(ipw)=smt(ipw)+samt(ipw,ita)-samt0(ipw,ita)
187          hmt(ipw)=hmt(ipw)+hamt(ipw,ita)-hamt0(ipw,ita)
188    70  continue
189          call swatch(61,2,'do 70   ')
190          call swatch(62,1,'do 76   ')
191      do 76 je=1,ne
192          smm(je,ie)=0.d0
193          hmm(je,ie)=0.d0
194      do 78 ipw=1,npw(k)
195          smm(je,ie)=smm(je,ie)
196          &           +conjg(cnn(ipw,je,k,is))*smt(ipw)
197          hmm(je,ie)=hmm(je,ie)
198          &           +conjg(cnn(ipw,je,k,is))*hmt(ipw)
199    78  continue
200    76  continue
201      hexp=hmm(je,ie)
202      sexp=smm(je,ie)
203      eps(ie,k,is)=hexp/sexp
204  ccccc  write(6,'(215,4d15.5)') k, ie, hmm(je,ie), smm(je,ie)
205          call swatch(62,2,'do 76   ')
206          call swatch(63,1,'do 80   ')
207      do 80 ipw=1,npw(k)
208          fon(ipw,ie,k,is)=-hmt(ipw)+eps(ie,k,is)*smt(ipw)
209    80  continue
210          call swatch(63,2,'do 80   ')
211          call swatch(64,1,'do 82   ')
212      do 82 ita=1,nats
213          do 84 ipw=1,npw(k)
214              l=indpw(1,ipw,k)
215              m=indpw(2,ipw,k)
216              n=indpw(3,ipw,k)
217              lgv=lstgv(l,m,n)
218              cmt =dimag(conjg(cnn(ipw,ie,k,is))
219          &           *(hamt(ipw,ita)-samt(ipw,ita)*eps(ie,k,is)))
220              cmt0=dimag(conjg(cnn(ipw,ie,k,is))
221          &           *(hamt0(ipw,ita)-samt0(ipw,ita)*eps(ie,k,is)))
222              fm(1,ita)=fm(1,ita)-2.d0*weit*cmt *l
223              fm(2,ita)=fm(2,ita)-2.d0*weit*cmt *m
224              fm(3,ita)=fm(3,ita)-2.d0*weit*cmt *n
225              fm0(1,ita)=fm0(1,ita)-2.d0*weit*cmt0*l
226              fm0(2,ita)=fm0(2,ita)-2.d0*weit*cmt0*m
227              fm0(3,ita)=fm0(3,ita)-2.d0*weit*cmt0*n
228    84  continue
229    82  continue
230          call swatch(64,2,'do 82   ')
231    22  continue
232  ccc  call cmatpr('hmm     ',hmm,1)
233  ccc  call cmatpr('smm     ',smm,1)
234 c diagonalize and rotate cnn
235      if(ldiag.gt.0) then
236          if(mod(loop,ldiag).eq.0) then
237              ij=0
238              do 26 ie=1,ne
239              do 26 je=1,ie
240                  ij=ij+1
241                  hr(ij)=0.5d0*(dreal(hmm(ie,je))+dreal(hmm(je,ie)))
242                  hi(ij)=0.5d0*(dimag(hmm(ie,je))-dimag(hmm(je,ie)))
243                  or(ij)=0.5d0*(dreal(smm(ie,je))+dreal(smm(je,ie)))
244                  oi(ij)=0.5d0*(dimag(smm(ie,je))-dimag(smm(je,ie)))
245    26  continue
246          call diagon(nemax,ne,ne,hr,hi,or,oi,

```

```

247      &           eps(1,k,is),zr,zi,work)
248      do 28 ie=1,ne
249      do 28 je=1,ne
250      zmm(ie,je)=dcomplex(zr(ie,je),zi(ie,je))
251      28   continue
252      call rotele(is,k,cnn,fcn,zmm)
253      endif
254      endif
255      21 continue
256      20 continue
257 c make the atomic forces into those in the cartesian units
258      do 90 ita=1,nats
259          f1=fm (1, ita)
260          f2=fm (2, ita)
261          f3=fm (3, ita)
262          fm (1, ita)=rcmt(1,1)*f1+rcmt(1,2)*f2+rcmt(1,3)*f3
263          fm (2, ita)=rcmt(2,1)*f1+rcmt(2,2)*f2+rcmt(2,3)*f3
264          fm (3, ita)=rcmt(3,1)*f1+rcmt(3,2)*f2+rcmt(3,3)*f3
265      90 continue
266      do 92 ita=1,nats
267          f1=fm0(1, ita)
268          f2=fm0(2, ita)
269          f3=fm0(3, ita)
270          fm0(1, ita)=rcmt(1,1)*f1+rcmt(1,2)*f2+rcmt(1,3)*f3
271          fm0(2, ita)=rcmt(2,1)*f1+rcmt(2,2)*f2+rcmt(2,3)*f3
272          fm0(3, ita)=rcmt(3,1)*f1+rcmt(3,2)*f2+rcmt(3,3)*f3
273      92 continue
274 c symmetry consideration
275      do 86 ita=1,nats
276          do 88 ks=1,nsym
277              ksi=invrot(ks)
278              jta=itarot(ks, ita)
279              fom (1, ita)=fom (1, ita)+drot(1,1,ksi)*fm (1, jta)
280              &                     +drot(1,2,ksi)*fm (2, jta)
281              &                     +drot(1,3,ksi)*fm (3, jta)
282              fom (2, ita)=fom (2, ita)+drot(2,1,ksi)*fm (1, jta)
283              &                     +drot(2,2,ksi)*fm (2, jta)
284              &                     +drot(2,3,ksi)*fm (3, jta)
285              fom (3, ita)=fom (3, ita)+drot(3,1,ksi)*fm (1, jta)
286              &                     +drot(3,2,ksi)*fm (2, jta)
287              &                     +drot(3,3,ksi)*fm (3, jta)
288              fom0(1, ita)=fom0(1, ita)+drot(1,1,ksi)*fm0(1, jta)
289              &                     +drot(1,2,ksi)*fm0(2, jta)
290              &                     +drot(1,3,ksi)*fm0(3, jta)
291              fom0(2, ita)=fom0(2, ita)+drot(2,1,ksi)*fm0(1, jta)
292              &                     +drot(2,2,ksi)*fm0(2, jta)
293              &                     +drot(2,3,ksi)*fm0(3, jta)
294              fom0(3, ita)=fom0(3, ita)+drot(3,1,ksi)*fm0(1, jta)
295              &                     +drot(3,2,ksi)*fm0(2, jta)
296              &                     +drot(3,3,ksi)*fm0(3, jta)
297      88 continue
298      fom (1, ita)=fom (1, ita)/nsym
299      fom (2, ita)=fom (2, ita)/nsym
300      fom (3, ita)=fom (3, ita)/nsym
301      fom0(1, ita)=fom0(1, ita)/nsym
302      fom0(2, ita)=fom0(2, ita)/nsym
303      fom0(3, ita)=fom0(3, ita)/nsym
304      86 continue
305 c add all atomic forces to make the total forces
306      do 94 ita=1,nats
307          foa(1, ita)=fog(1, ita)+fom(1, ita)-fom0(1, ita)
308          foa(2, ita)=fog(2, ita)+fom(2, ita)-fom0(2, ita)
309          foa(3, ita)=fog(3, ita)+fom(3, ita)-fom0(3, ita)
310      94 continue
311 c      if(lprint.gt.0) then

```

```

313      if(mod(loop,lprint).eq.0) then
314          call forcpr(foa,fog,fom,fom0)
315      endif
316      endif
317      call writ24(foa,fog,fom,fom0)
318 c make the total atomic forces into those in lattice vector units
319      do 96 ita=1,nats
320          fx=foa(1,ita)
321          fy=foa(2,ita)
322          fz=foa(3,ita)
323          foa(1,ita)=(rcmt(1,1)*fx+rcmt(2,1)*fy+rcmt(3,1)*fz)/tpi
324          foa(2,ita)=(rcmt(1,2)*fx+rcmt(2,2)*fy+rcmt(3,2)*fz)/tpi
325          foa(3,ita)=(rcmt(1,3)*fx+rcmt(2,3)*fy+rcmt(3,3)*fz)/tpi
326 96 continue
327 c
328      return
329      end

```

(orthon)

```

1 c---- orthon
2 c
3     subroutine orthon(nspin,cnn,loop)
4     implicit real*8 (a-h,o-z)
5     parameter(nbase=1499)
6     parameter(nemax=50, nkmax=36)
7     parameter(ntype=4, nsite=7, mstype=2, nrad=401, nrad0=61)
8     parameter(nsp=1)
9     parameter(lup=2, lmax=(lup+1)**2)
10    parameter(nloop=500)
11 c
12    complex*16 cnn(nbase, nemax, nkmax, nsp)
13 c
14    complex*16 smm(nemax, nemax)
15    complex*16 smt(nbase)
16    complex*16 samt(nbase, nsite)
17    complex*16 samt0(nbase, nsite)
18    complex*16 ss(nbase)
19    complex*16 ylmk, exkr, p, q, t, u
20    complex*16 aan, ban, wt
21 c
22    real*8 rcmt(3,3)
23 c
24    common/cbasis/brcmt(3,3)
25    common/ckpvec/ak(3, nkmax), wk(nkmax), nkpt
26    common/csizes/npw(nkmax), ne
27    common/cdelta/dt(nloop), dta(nloop), mode
28    common/catoms/zval(ntype), idmsh(ntype), natom(ntype), lmax, ntp
29    common/cradii/radmt(mstype), xstep(mstype),
30    & nrd(mstype), nrd0(mstype), mstp
31    common/cradw0/rval0(nrad0, mstype), rwei0(nrad0, mstype)
32    common/casfun/pf(0:lup, ntype, nsp), dpf(0:lup, ntype, nsp),
33    & pef(0:lup, ntype, nsp), dpef(0:lup, ntype, nsp),
34    & penorm(0:lup, ntype, nsp), wronsk(0:lup, ntype, nsp)
35    common/catome/ezr(0:lup, ntype, nsp)
36    common/cwork1/ylmk(nbase, lmax), exkr(nbase, nsite),
37    & p(lmax, nsite), q(lmax, nsite),
38    & t(nrad, lmax, nsite), u(nrad, lmax, nsite),
39    & aln(nbase, 0:lup, ntype, nsp),
40    & bln(nbase, 0:lup, ntype, nsp),
41    & sjlnr(nbase, nrad, 0:lup, mstype)
42    common/cprint/lprint, ngpr, incr, incr0
43 c
44    if(dt(loop).le.0.d0) return
45 c

```

```

46      tpi=8.d0*atan(1.d0)
47      fpi=tpi+tpi
48      vol=abs(deter3(brmt))
49      vi=1.d0/vol
50      c
51      do 20 k=1,nkpt
52          call mkylmk(k,ylmk)
53          call mkekxr(k,exkr)
54          call mkabin(nspin,k,aln,bln)
55          call mkjln0(k,sjlnr)
56          do 21 is=1,nspin
57          do 22 ie=1,ne
58              call mkpqt0(is,k,ie,cnn,ylmk,exkr,aln,bln,sjlnr,p,q,t)
59              do 30 ita=1,nsite
60              do 30 ipw=1,nbase
61                  samt(ipw,ita)=0.d0
62                  samt0(ipw,ita)=0.d0
63      30      continue
64          ita=0
65          do 40 it=1,ntp
66          do 40 ia=1,natom(it)
67              ita=ita+1
68              im=idmsh(it)
69              lm=0
70              do 42 l=0,imax
71              do 42 m=-l,l
72                  lm=lm+1
73                  do 44 ipw=1,npw(k)
74                      aan=conjg(ylmk(ipw,lm))*exkr(ipw,ita)*aln(ipw,l,it,is)
75                      ban=conjg(ylmk(ipw,lm))*exkr(ipw,ita)*bln(ipw,l,it,is)
76                      samt(ipw,ita)=samt(ipw,ita)
77                      & +conjg(aan)*p(im,ita)
78                      & +conjg(ban)*q(lm,ita)*penorm(l,it,is)
79      44      continue
80          do 50 ipw=1,npw(k)
81              ss(ipw)=ylmk(ipw,lm)*conjg(exkr(ipw,ita))
82      50      continue
83          do 52 ir=1,nrd0(im)
84              wt=t(ir,lm,ita)*rwei0(ir,im)
85          do 54 ipw=1,npw(k)
86              samt0(ipw,ita)=samt0(ipw,ita)
87              & +ss(ipw)*sjlnr(ipw,ir,l,im)*wt
88      54      continue
89      52      continue
90      42      continue
91      40      continue
92          ita=0
93          do 60 it=1,ntp
94          do 60 ia=1,natom(it)
95              ita=ita+1
96              do 62 ipw=1,npw(k)
97                  samt(ipw,ita)=samt(ipw,ita)*fpi*vi
98                  samt0(ipw,ita)=samt0(ipw,ita)*fpi*vi
99      62      continue
100     60      continue
101     do 70 ipw=1,npw(k)
102         smt(ipw)=cnn(ipw,ie,k,is)
103     70      continue
104         ita=0
105         do 72 it=1,ntp
106         do 72 ia=1,natom(it)
107             ita=ita+1
108             do 72 ipw=1,npw(k)
109                 smt(ipw)=smt(ipw)+samt(ipw,ita)-samt0(ipw,ita)
110     72      continue
111     do 76 ie=1,ne

```

```
112      smm(je, ie)=0. d0
113      do 78 ipw=1,npw(k)
114          smm(je, ie)=smm(je, ie)+conj(cnn(ipw, je, k, is))*smt(ipw)
115      78      continue
116      76      continue
117      22      continue
118      c
119      c      if(lprint.gt.0) then
120      c          if(mod(loop, lprint).eq.0) then
121      c              call cmatpr('smm      ', smm, 1)
122      c          endif
123      c      endif
124      c
125      call schmit(is, k, cnn, smm)
126      21      continue
127      c
128      20      continue
129      return
130      end
```