

JAERI-Data/Code  
2000-019



JP0050365



# 第一原理分子動力学法プログラムの並列化

2000年3月

渡部 弘・小林一昭\*・新井正男\*

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。  
入手の問合せは、日本原子力研究所研究情報部研究情報課（〒319-1195 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（〒319-1195 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, 319-1195, Japan.

© Japan Atomic Energy Research Institute, 2000

編集兼発行　　日本原子力研究所

## 第一原理分子動力学法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・小林 一昭\*・新井 正男\*

(2000年 2月9日受理)

第一原理分子動力学法プログラムの並列化について報告する。並列化したプログラムは2本である。1つは伝統的な第一原理分子動力学法プログラム、もう一つは近年、高速化という観点から注目されている実空間分子動力学法プログラムである。ターゲットマシンは SX-4 であり、並列化手法としては共有メモリ型並列化手法を採用した。並列化の結果、第一原理分子動力学法プログラムは8並列で3.7倍、実空間分子動力学法プログラムでは8並列で3.9倍の高速化を達成した。

## Parallelization for First Principles Molecular Dynamics Programs

Hiroshi WATANABE, Kazuaki KOBAYASHI\* and Masao ARAI\*

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

(Received February 9, 2000)

In this report we study parallelization for two First Principles molecular dynamics programs. One is a traditional First Principles molecular dynamics program, and the other is a real-space molecular dynamics program which attracts an attention recently because of its high speed. The target machine is NEC SX-4, with the parallelization method adopted for shared memory machines. It is shown that 3.7 times acceleration is achieved with 8 CPU parallelization for the traditional First Principles molecular dynamics program and 3.9 times acceleration with 8 CPU parallelization for the real-space molecular dynamics program.

Keywords: First Principles, Molecular Dynamics, Real-Space, SX-4, Parallelization

---

\*National Institute for Research in Inorganic Materials

## 目 次

1 はじめに .....	1
2 第一原理分子動力学法の概略 .....	2
3 第一原理分子動力学法プログラムの並列化 .....	5
3.1 プログラムの性能解析 .....	5
3.2 SX-4 における共有メモリ型並列化 .....	7
3.3 並列化作業 .....	9
4 並列化 revpb の性能測定と考察 .....	13
5 実空間分子動力学法プログラムの並列化 .....	16
5.1 プログラムの性能解析 .....	16
6 並列化 realmesh の性能測定と考察 .....	19
7 おわりに .....	21
謝 辞 .....	21
参考文献 .....	22
付録 .....	23

## Contents

<b>1</b>	<b>Introduction.....</b>	<b>1</b>
<b>2</b>	<b>Survey of First Principles Molecular Dynamics.....</b>	<b>2</b>
<b>3</b>	<b>Parallelization of First Principles Molecular Dynamics .....</b>	<b>5</b>
3.1	Investigation of Computing Cost .....	5
3.2	Parallelization Method of SX-4 .....	7
3.3	Parallelization .....	9
<b>4</b>	<b>Performance Evaluation and Discussion .....</b>	<b>13</b>
<b>5</b>	<b>Parallelization of Real-Space Molecular Dynamics Program.....</b>	<b>16</b>
5.1	Investigation of Computing Cost .....	16
<b>6</b>	<b>Performance Evaluation and Discussion .....</b>	<b>19</b>
<b>7</b>	<b>Conclusion .....</b>	<b>21</b>
<b>Acknowledgements .....</b>		<b>21</b>
<b>References .....</b>		<b>22</b>
<b>Appendix.....</b>		<b>23</b>

## 1 はじめに

第一原理分子動力学法とは、量子力学に基づき分子内の電子状態を求め、それによる原子間ポテンシャルから多数の原子の動力学を計算する手法である。この手法により数百原子から構成される分子の安定構造や欠陥、表面構造を、理論的かつ高精度で決定することができる。さらに最近では、触媒反応、固体表面の分子かい離、分子の励起状態の構造決定等にも適用されている。

第一原理分子動力学法は理論的計算であるから、当然、爆発性や毒性のある分子や非常に寿命の短い分子の性質を詳細に調べることができる。また密度汎関数理論という、それ自体にはなんの近似も含まない厳密な理論を基盤とするので、実験値の参照や人為的なパラメータの導入を必要とせずに、精度の高い計算が可能である。これらの特徴により、第一原理分子動力学法は物性研究や各種新材料設計、分子設計の分野で広く利用されている。

もちろん第一原理分子動力学法にも種々の課題がある。特に計算時間の問題は深刻である。同じ第一原理計算である分子軌道法よりはるかに改善されているとはいえ、第一原理分子動力学法の計算量は膨大であり、マイクロ秒単位の長い時間スケールでの原子の運動を追跡することや、数千個以上の原子を含む系を解析することは、現在のところ、非常に困難である。そのため高速な第一原理分子動力学法プログラムの開発は急務である。

以上の背景の下、我々は第一原理分子動力学法プログラムの並列化を行い、高速化を試みた。单一CPUによる性能向上が頭打ちになっている今日、並列化はプログラム高速化における最有力手段である。対象に選んだプログラムは、伝統的な第一原理分子動力学法プログラムと、近年、並列化しやすいという観点から注目されている実空間分子動力学法プログラムの2本である。両プログラムとも無機材質研究所で開発され、実際に物性研究や分子設計の研究現場で利用されているプログラムである。またターゲットマシンとしては、並列化作業が容易であることと筆者等がアクセスし易いことから、NEC製SX-4を採用した。

本報告書の構成は次の通りである。第2章で第一原理分子動力学法の概略を、分子軌道法や他の分子動力学法と対比しながら説明する。あわせて、今回の並列化の対象となった第一原理分子動力学法プログラムと実空間分子動力学法プログラムの位置付けもここで明らかにする。第3章で第一原理分子動力学法プログラムの性能解析及び並列化方針の決定を行う。第3章ではさらに、SX-4特有の共有メモリ型並列化手法であるマイクロタスク並列化についても説明する。マイクロタスク並列化は、MPIに代表されるメッセージパッシング方式の並列化よりもはるかに簡単で性能の出やすい並列化手法である。第4章で実際に性能測定を行い、計測結果について考察する。第5章と第6章では、今回並列化したもう一つのプログラムである実空間分子動力学法プログラムについて、同様の議論を行う。

## 2 第一原理分子動力学法の概略

第一原理分子動力学法は分子軌道法や古典的分子動力学法とともに、計算化学の分野で物性研究や物質・材料設計の際に有力な計算手法として、広く利用されている。これらの手法を整理すると次のようになる[1]。

表 1 計算化学の各手法の特徴

名称	基礎方程式	経験的／非経験的	原子数	計算精度
分子軌道法	Schrödinger方程式	非経験的	数十原子	◎
第一原理 分子動力学法	Kohn-Sham方程式	非経験的	数百原子	○
古典的 分子動力学法	運動方程式	経験的	数千原子から 数万原子	△

分子軌道法と第一原理分子動力学法は、実験値等の入力を必要としないという意味で非経験的であり、原理的には利用者のスキルによらずに精度の高いシミュレーションが可能である。非経験的手法の著しい長所としては、分子内の電子状態もシミュレートするため化学反応や物質の相転移といった現象が解析可能という点があげられる。これらの現象は古典的分子動力学法では全く扱えなかつたものである。

分子軌道法は量子力学で最も基本的な Schrödinger 方程式を数値的に解く手法である。シミュレーションの精度は3手法の中で最も高く、その計算精度は実験値に匹敵する程である。しかし、原理的には各原子に属する全電子の波動関数を計算しなければならず、計算量は非常に膨大である。そのため計算可能な原子数は数十原子に留まっている。

一方、古典的分子動力学法は、原子間のポテンシャルを人工的に仮定して原子の運動を計算する手法である。基礎方程式は古典的な運動方程式であり、計算精度はひとえに仮定される原子間ポテンシャルの質に依存する。そのため適切なポテンシャルが採用されれば精度の良い結果が得られるが、そうでないと全く現実と異なるシミュレーションとなる恐れもある。この意味で古典的分子動力学法プログラムの計算精度は、利用者のスキルに依存するといえる。もちろんその分、計算量は非経験的手法に比べ著しく軽減され、はるかに大規模な計算が可能である。

第一原理分子動力学法は古典的分子動力学法と同様、原子の運動を運動方程式により計算するが、原子間ポテンシャルをKohn-Sham方程式により非経験的に求める部分が異なる。この方程式は、基底状態の分子内電子の波動関数を決定する方程式であり、密度汎関数理論から導出される。その事情を以下に説明する([8][9])。

原子または分子内の電子密度  $n$  は、電子の波動関数  $\phi_i$  から以下の式で求められる。Nは計算対象となっている全電子数である。

$$n = \sum_{i=1}^N |\phi_i|^2$$

この式からもわかるように、電子の波動関数から電子密度を計算することは容易である。一方、電子密度から波動関数を求めることは、一般には不可能である。ところが1964年Kohn等は、基底状態における電子の波動関数は電子密度が分かれば決定可能であり、しかもその電子密度は、電子密度を変数とするエネルギー汎関数( = 電子のクーロンポテンシャル + 電子の運動エネルギー + 電子間相互作用エネルギー)を最小にすることを、厳密に証明した。これが密度汎関数理論である([2])。このエネルギー汎関数に変分原理を適用することにより、次のKohn-Sham方程式が導出される。

$$\{-(\hbar^2/2m)\nabla^2 + V_{\text{eff}}\}\Phi_i = \varepsilon_i \Phi_i$$

$$V_{\text{eff}} = V_{\text{nucleus}} + V_{\text{electron}} + V_{\text{xc}}$$

ここで  $V_{\text{eff}}$  は有効ポテンシャルと呼ばれ、電子が分子中の原子核や他の電子から受けるポテンシャルである。有効ポテンシャル  $V_{\text{eff}}$  の第一項  $V_{\text{nucleus}}$  は各原子核からのクーロンポテンシャル、第二項  $V_{\text{electron}}$  は電子平均密度によるクーロンポテンシャルである。これらの項はそれぞれの計算式により容易に計算できる。一方、第三項  $V_{\text{xc}}$  は交換相関ポテンシャルと呼ばれ、本来の多体効果は全てこの項に凝縮されている。この項は電子密度を変数として持ち、従って電子の波動関数から決定される。そのため、Kohn-Sham方程式は非線形方程式であり、反復的に自己無撞着になるように解く必要がある。

Kohn-Sham方程式は、みかけ上はハートリー・フォック近似による一体問題のSchrodinger方程式に類似している。しかしこの式自体には、いかなる近似も含まない厳密なものであることに注意する必要がある。本来、高度な多体問題である電子状態計算を、見かけ上とはいえ一体問題に帰着できるメリットは大きく、計算量が著しく軽減される。それゆえに第一原理分子動力学法は分子軌道法よりも多くの原子を扱うことが可能である。このことは1原子あたりの計算量を考えると明らかである。素朴な分子軌道法では多体問題として電子の個数分のSchrödinger方程式を解かなければならぬが、第一原理分子動力学法では1個のKohn-Sham方程式を解けば十分である。

実際には第一原理分子動力学法でも多体問題が完全に解決された訳ではなく、電子の多体効果は交換相関ポテンシャル項  $V_{\text{xc}}$  に残っている。 $V_{\text{xc}}$  の正確な表示式は分かっていないため、この項に何らかの近似が必要となる。従来は局所密度近似(LDA)と呼ばれる最も簡単な近似が採用されてきたが、近年では、より精度の高い一般化密度勾配近似(GGA)も採用されるようになってきた[3]。GGAの採用に伴い、第一原理分子動力学法の計算精度は飛躍的に向上し、シミュレーションの信頼性は分子軌道法と同程度にまで高められている。

以上のことから、第一原理分子動力学法プログラムは次のようなアルゴリズムで構成される。

- (1) 原子配置及び電子状態の初期値を設定する。
- (2) 与えられた原子配置の下で、Kohn-Sham方程式を自己無撞着になるように解く。
- (3) (2)で得られた電子の波動関数から、各原子のポテンシャルを決定する。
- (4) (3)のポテンシャルから原子の運動方程式を解く。
- (5) (2)から(4)のステップを収束するまで繰り返す。

(2)のステップでKohn-Sham方程式を数値的に解くには、極座標形式の方程式の波動関数をフーリエ級数展開し、その係数をFFTにより求める手法が従来から用いられてきた。その理由は、分子の結晶構造が周期構造を持っておりフーリエ級数展開になじむこと、FFTの利用により少ない計算量で高精度の計算が可能であったことによる。

しかしこれらの理由は、現在では必ずしも著しい利点とはいえないくなっている。第一の周期構造については、近年の大きな研究対象である物質の表面や界面では、周期境界条件が適用できない。また第二の理由であるFFTの優位性も、現在の計算機では高速性を保証するものではない。なぜならFFTはベクトル型の計算に適さず、しかも並列化也非常に困難だからである。

上記のことから、近年ではFFTを用いずに、オリジナルのKohn-Sham方程式を場の問題とみなして、実空間上で離散化して数値的に解くアプローチが注目されている。この手法は実空間分子動力学法と呼ばれている。実空間分子動力学法は、従来の第一原理分子動力学法に比べ精度的には不利であるが、物質の表面構造や界面構造を問題なく扱える、ベクトル計算機や並列計算機上では非常に高速な計算が可能、プログラミングが容易という利点から、注目を集めている。

本論文では、伝統的なFFTによる第一原理分子動力学法プログラムであるrevpbと、実空間分子動力学法プログラムであるrealmeshに、共有メモリ型の並列化を行い、高速化を図っている。

### 3 第一原理分子動力学法プログラムの並列化

#### 3.1 プログラムの性能解析

本章と次章では、伝統的な第一原理分子動力学法プログラムrevpbの並列化について述べる。本プログラムのアルゴリズムは第2章で述べたことが基本となっている。アルゴリズムの詳細は参考文献[6]及び付録Aを参照されたい。

既存の逐次型のプログラムを並列化する場合には、まずそのプログラムのどの部分が計算時間を多く消費しているかを特定する必要がある。実際に研究開発の現場で利用されているプログラムの中には、数千行から数万行を超えるものが少なくない。今回並列化するプログラムrevpbも1万行を超える大規模なものであり、この全てを詳細に調査し可能な部分を並列化するのは、作業工数の面から見ても非現実的である。むしろプログラム各部分の消費時間の調査を行い、コストの大きい部分から並列化を行う方が効率的である。

まずプログラム全体の性能解析を行う。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計測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フロップス値を非常に容易に求めることができる。実際にオリジナルのrevpbに対し、プログラム性能解析を行った結果が次である。

***** Program Information *****		
Real Time (sec)	:	984. 340797
User Time (sec)	:	974. 182738
Sys Time (sec)	:	6. 138198
Vector Time (sec)	:	903. 956929
Inst. Count	:	28830100831.
V. Inst. Count	:	11174985137.
V. Element Count	:	1896620566250.
FLOP Count	:	794337429180.
MOPS	:	1965. 006776
MFLOPS(フロップス値)	:	815. 388528
VLEN(平均ベクトル長)	:	169. 720187
V. Op. Ratio(%) (ベクトル化率)	:	99. 077713
Memory Size (MB)	:	52. 031250
MIPS	:	29. 594140
I-Cache (sec)	:	9. 821021
O-Cache (sec)	:	20. 788573
Bank (sec)	:	150. 695796
real	16:25.09	
user	16:14.18	
sys	6.43	

図 1 プログラム性能情報(逐次版)

上記の結果で目に付くのは、ベクトル化率が99%を超えていてもかかわらず、フロップス値(MFLOPS)が約800MFLOPSという点である。SX-4の1CPUの最大フロップス値は2GFLOPSなので、この値はあまり満足すべき値とはいえない。これは主要なDOループのベクトル化はされているが、ベクトルレパインが有効に活用されていないことを意味する。その原因として、バンクコンフリクトが大きいことと平均ベクトル長が短いことが考えられる。これらは本プログラム中で最も計算時間を

消費しているFFTに起因する。FFTはベクトル長が計算途中で変化し、ベクトル長の短いDOループが頻出する。しかもそのループの間隔が2の巾乗間隔なので、バンクコンフリクトを起こしやすい。これらの問題を解決するには、FFTルーチンの大幅な改良と、それに伴うプログラム全体のデータ構造の変更を必要とする。これは非常に困難な作業と予想される。

次にサブルーチン単位の経過時間(正確には正味経過時間、定義については後述)を計測してみる。計測には、日本原子力研究所計算科学技術推進センターが開発しているプログラム性能解析ツールKMtool90を利用した[4]。

表 2 revpb 各ルーチンの正味経過時間

サブルーチン名	プログラム実行時間に対する比率
msd	13.86%
mfftc5	11.12%
mfft b5	10.58%
mfftov	10.36%
mfftc7	6.37%

この結果を見ると、最もCPU時間を消費しているサブルーチンmsdでも全体の14%弱となっており、非常に分散している。従って、各ルーチンをそれぞれチューニングして高速化を図るのは非効率的である。なお、サブルーチンmsdは原子内の電子状態を計算するルーチンで、本プログラムの中核部分である。また、表の中のmfftxxxというルーチンは全てFFTルーチン群である([10][11])。第2章でも述べたように、FFTルーチンを並列化することは非常に困難であり、高度な数値計算技術を要する。従って安易に並列化を試みるべきではない。

そこで我々は、個々のルーチンをベクトル化等の最適化により高速化することは断念し、できるだけ上位のルーチンを並列化することにより高速化を図ることにした。そのためには、まず全てのルーチンの全経過時間を計測し、どのルーチンを並列化すべきかを特定しなければならない。ここで、正味経過時間と全経過時間の違いを図2で説明する。サブルーチンsub0が別のサブルーチンsub1をcallしている場合を考える。sub0の全経過時間には、実際にはsub1が実行されている時間も含まれる。一方、sub0の正味経過時間にはsub1の時間は含まれない。これからベクトル化による高速化作業には正味経過時間が、並列化では全経過時間が重要であることが分かる。

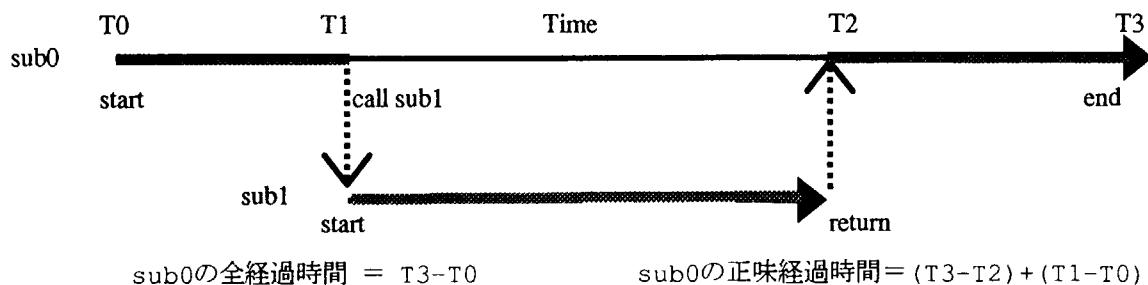


図 2 全経過時間と正味経過時間

やはり前述の性能解析ツールKMtool90を用いて計測したルーチンの全経過時間は次の通りである。

表 3 revpb 各ルーチンの全経過時間

サブルーチン名	プログラム実行時間に対する比率
MAIN	100.00%
c3fft	76.59%
msd	64.34%
chaver	28.43%
mfftiv	27.08%

MAINは本プログラムのメインルーチンなので全経過時間の比率は、当然100%である。またc3fftは3次元用のFFTルーチン群のドライバルーチンであり、ここでデータ数の素因数分解を行い、適切な基底数のFFTルーチンを呼び出している。これらのルーチンを調査したところ、そのどちらにも並列性を有するブロックはなく並列化は不可能であった。一方、電子状態計算ルーチンmsdと電子密度計算ルーチンchaverは、合計でプログラム全体の93%の時間を消費し、しかもその中に高い並列性を有するDOループを持っていることがわかった。そこでこれらのルーチンを並列化し、プログラムの性能向上を図ることにする。

CPU消費時間による性能解析で大きな計測値を示していたFFTルーチン群は全て、このDOループ中からサブルーチンとしてcallされている。従って、上記の方針はFFTルーチン自体を並列化することは避け、FFTルーチンをcallする側を並列化することによりFFTルーチンを並列実行させることに相当する。

### 3.2 SX-4における共有メモリ型並列化

並列計算機は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。共有メモリ型計算機は1個の大容量のメモリを複数個のCPUが共有する方式であり、メモリ中のデータは全てのCPUから参照や修正が可能である。一方、分散メモリ型計算機では、1つのCPUと比較的小容量のメモリからなるPE(Processing Element)が高速なネットワークで結合されている。分散メモリ型の計算機ではあるPEが他のPEのデータにアクセスするには特別な手順が必要となり、その速度も同一PE内のメモリアクセスに比べ非常に遅い。

共有メモリ型の並列化はプログラミングが比較的容易で、自動並列化コンパイラがサポートされている場合も多い。しかし、多数のCPUを持つ共有メモリ型並列計算機を開発するのは困難であり、一般に32CPU程度の共有メモリ型並列計算機が限界といわれている。またメモリアクセス競合が起きやすく、CPU数の増加に比例した性能向上が達成しにくいという欠点がある。

一方、分散メモリ型並列計算機では、数100から数1000PEを有する大規模な並列計算機も既に実用化されており、カタログ性能値は共有メモリ型のそれをはるかに凌駕する。また最近では、ワーカステーションクラスタやPCクラスタに見られるように、比較的安価な計算機をネットワークで連結し、

コストパフォーマンスに優れた分散メモリ型並列計算機を開発する試みもなされている。ところが、分散メモリ型プログラミングは作業量が大きく、大規模で複雑なプログラムに分散メモリ型の並列化をすることは多大な労力と時間を必要とする。

このようにそれぞれの方式には一長一短があり、どちらの方式が優れているとは、一概にはいえない。一般的には、新たに並列化プログラムを開発する場合や超高速な計算を必要とする場合には分散メモリ型の並列化が、既存の逐次型プログラムを高速化する場合には共有メモリ型の並列化が適しているといえる。また最近では分散共有型と呼ばれる、両方の方式を兼ね備えた方式も一般的になっている。この方式は共有メモリ型の並列計算機を1つのPEとし、それを高速なネットワークで接続する方式である。

今回のターゲットマシンであるSX-4も分散共有型の並列計算機であり、分散メモリ型と共有メモリ型の両方式のプログラミング手法がサポートされている。具体的には、分散メモリ型並列化手法としてはMPIによる並列化プログラミング手法が、共有メモリ型並列化手法としてはマイクロタスク並列化と呼ばれる並列化指示行による並列化手法が利用できる[5]。

本稿で述べる並列化作業では、以下の理由からマイクロタスク並列化手法を採用した。

- (1)並列化に要する作業量が、MPIによる並列化に比べてはるかに少ないこと。
- (2)並列化指示行は他の計算機ではコメントとみなされるので、逐次型の計算機にはそのまま移植できること。
- (3)マルチプロセッサのPCやワークステーションのように、比較的安価な共有メモリ型並列計算機が市場に出てきたこと。

以下ではSX-4のマイクロタスク並列化手法について詳述する。共有メモリ型の並列化は、作業量自体は比較的少ないが、概念的には分散メモリ型の並列化より難しい面がある。それは並列実行される範囲と変数の取り扱い方法がやや複雑で、しかも計算機種によって異なるからである。

SX-4のマイクロタスクによる並列化では、並列実行される範囲は、実際に並列化指示行が挿入されたサブルーチン(並列化ルーチン)全体である。その様子を図3で説明する。並列化指示行のない上位ルーチンはCPU0でのみ実行される。このルーチンが並列化ルーチンsubparをcallすると、指定された個数のCPUが確保され、各CPUでsubparが最初から実行される。すなわち、並列化ルーチンsubpar中の並列化指示行付きのDOループは各CPUで並列実行され、そのDOループの前後の部分は各CPUで冗長に実行される。このような仕様は並列化によるオーバーヘッドを少なくするためと考えられる。

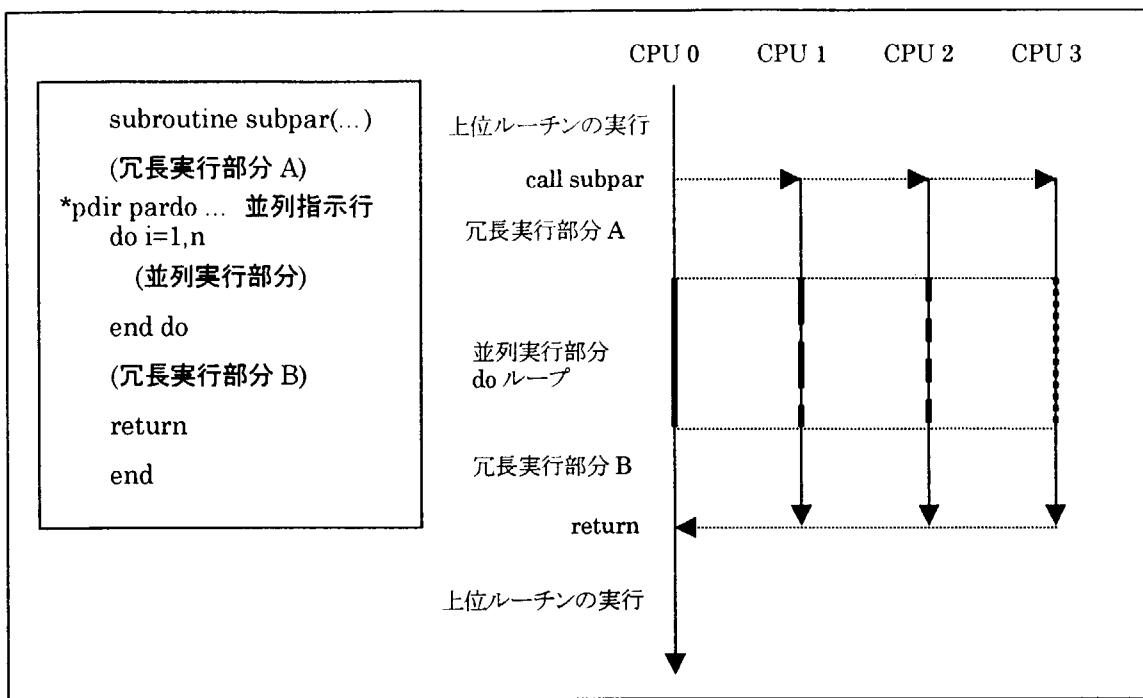


図 3 マイクロタスク実行の様子

共有メモリ型並列化では、変数はグローバル変数とローカル変数の2種類に分類される。グローバル変数はメモリ中に1つだけ確保され、複数のCPUから参照や修正が可能な変数である。一方、ローカル変数は各CPUに対応してCPU数分だけ確保される。各ローカル変数は異なる値を持ち、他のCPUからは変更も参照もできない。共有メモリ型の並列化の場合はこれらの変数の指定が問題となりやすい。例えば共有メモリ型並列化指示行の標準として注目されているOpen MPでは、基本的に変数の割り振りはユーザの責任であり、原則としてユーザが全ての変数の割り振りを行わなければならない。

SX-4マイクロタスク並列化では、以下のどれかの条件を満たす変数がグローバル変数となり、それ以外はすべてローカル変数として扱われる。

- (1)common文で指定された変数。
- (2)data文やsave文で指定された変数。
- (3)並列化ルーチン(並列化指示行を含むルーチン)の引数。

SX-4では、ローカル変数はプログラム実行中に必要になった時点で動的に確保され、必要なくなった時点で解放される。従って、サブルーチン内でのみ必要な変数はローカル変数として扱われ、そのサブルーチン終了後も必要な変数や全CPUからアクセス可能にすべき変数はグローバル変数として扱われると言えれば分かりやすい。

### 3.3 並列化作業

前節で述べたことを元に、サブルーチン `msd` と `chaver` の並列化作業を行った。並列化されたソースコードは付録Aに掲載されているので、詳細はそちらを参照されたい。並列化作業の概要は以

下の通りである。

- (1) オリジナルのプログラムではメモリ容量を節約するため、ワーク用の配列を common 文で宣言していた。しかしそれではワーク用配列はグローバル変数となり、各 CPU が自由にワーク用配列に書きこみを行い、正しい値が得られなくなる。そこで msd と chaver のワーク用配列は common 文から削除し、改めて dimension 文で宣言した。
- (2) サブルーチン msd 内の並列化 DO ループより前の部分は \*pdir serial～\*pdir endserial という並列化指示行で囲む。この指示行は CPU0 でのみの実行を意味する。これは冗長実行により、common 変数に全 CPU からの書きこみを防ぐためである。
- (3) サブルーチン msd 内の並列化対象 DO ループの直前に、並列化指示行 \*pdir pardo を挿入する。またその DO ループ中にエラーチェック用の stop 文があったが、並列化を妨げるのでこれを取り除いた。
- (4) 上記の DO ループ内から FFT のドライバールーチン c3fft が call され、さらに c3fft が多数の FFT ルーチン群を call している。ここで問題になるのは、これら FFT ルーチン群で使われている common 変数である。なぜなら、FFT ルーチン群で使用されている common 変数はルーチン間で共有されるためのものであり、グローバル変数として複数の CPU からアクセスされることは想定されていないからである。SX-4 ではこの問題を回避するため、local common 文という FORTRAN の独自拡張仕様が用意されている。これは各 CPU ごとに割り当てられる common のことである。これを用い、FFT ルーチン群の全ての common 文を local common 文に置き換えた。
- (5) 同様に FFT ルーチン群中の save 文も local common 文に置き換えた。
- (6) サブルーチン chaver において、並列化対象 DO ループの前の部分では、FFT のセットアップルーチンの呼び出しや種々の変数の初期化がなされている。この部分は msd の時と同様に、\*pdir serial～\*pdir endserial で囲み CPU0 でのみ実行させれば良いのだが、そうするためにはそれら全ての変数を調査し、グローバル変数とローカル変数に振り分ける作業が必要となる。これは時間のかかる作業である上に、誤りを起こしやすい。そこで我々は、並列化対象 DO ループを別ルーチンとして切り出し、その DO ループで使われている変数は全て引数として渡すこととした。これは機械的な作業なので誤りが少なく、引数として渡された変数はグローバル変数として扱われる所以、自動的に全 CPU から参照可能になるというメリットがある。
- (7) ただ一つだけ注意すべき点として、DO ループ内で合計値を求める部分がある。それを説明するため、1 から 100 までの合計値を求めるプログラム例を掲げる(図 4 参照)。これを逐次実行させると合計値 5050 が出力される。これをマイクロタスク並列化するのだが、初心者が犯しやすい誤りを図 5 に示す。このプログラムを 2 並列で実行すると、図にあるように、2 つの誤った数値が出力される。この値の和が正しい数値 5050 になることからも分かるように、この例で出力されたのは、各 CPU 内での合計値、すなわち部分和である。正しいマイクロタスク並列化例は図 6 のようになる。

```

program test0
integer sum
sum = 0
do i=1,100
    sum = sum + i
end do
write(6,*) ' sum = ',sum
end

```

図 4 総和計算(逐次版)

```

program test1
integer sum
sum = 0
*pd़ir pardo
do i=1,100
    sum = sum + i
end do
write(6,*) ' sum = ',sum
end

```

(2 並列での実行結果)  
sum = 2551  
sum = 2499

図 5 総和計算の誤った並列化例

```

program test2
integer sum,sum_local      ... sum_local はローカル変数
save   sum                  ... sum はグローバル変数
sum_local = 0               ... ローカル変数を初期化
*pd़ir pardo                ... DO ループを並列実行
do i=1,100
    sum_local = sum_local + i     ... 各 CPU 毎の部分和を計算
end do
*pd़ir serial              ... CPU0 でのみ実行する
    sum = 0                     ... グローバル変数を初期化
*pd़ir endserial
*pd़ir critical             ... 全 CPU で排他的に実行
    sum = sum + sum_local      ... グローバル変数に部分和を加算
*pd़ir endcritical
*pd़ir serial
    write(6,*) ' sum = ',sum    ... CPU0 からのみ出力
*pd़ir endserial
end

```

(2 並列での実行例)  
sum = 5050

図 6 総和計算(マイクロタスク並列化版)

図6でグローバル変数に部分和を加算する部分を \*pdir critical～\*pdir endcriticalで囲んでいる。これは排他的実行を表し、あるCPUが加算している時は他のCPUは加算できないことを意味する(図7参照)。この指示行がないと複数のCPUが同時に一つのグローバル変数に加算を行い、正しい値が得られない場合があるので、この指示行は必須である。

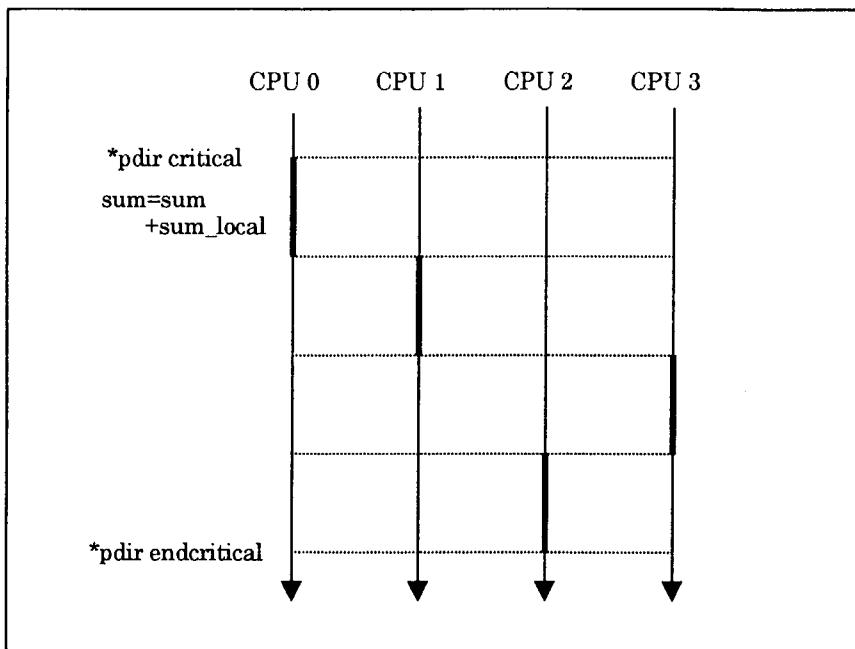


図 7 排他的実行の様子

(8)コンパイル方法は並列化指示行を有効にするオプション(-P multi)を用いる。

*> f77 -P multi \*.f*

(9)確保するCPU数を環境変数 F\_RSVTASK で指定した後、実行する。

*> setenv F\_RSVTASK 4*

*> a.out*

#### 4 並列化 revpb の性能測定と考察

並列化 revpb の性能測定を行ったところ、以下のような結果が得られた。なお、計測結果を早く得るために、反復数を 200 回から 40 回に制限している。

表 4 計測時間と加速率

	1CPU	2CPU	4CPU	8CPU
時間(秒)	213.54	119.38	79.73	57.11
加速率(測定値)	1.000	1.789	2.678	3.739
加速率(理想値)	1.000	1.744	2.776	3.943

表 5 並列化ルーチンの加速率

	1CPU	2CPU	加速率
msd	125.82sec	62.66sec	2.01
chaver	56.40sec	28.69sec	1.97
合計	182.22sec	91.35sec	1.99

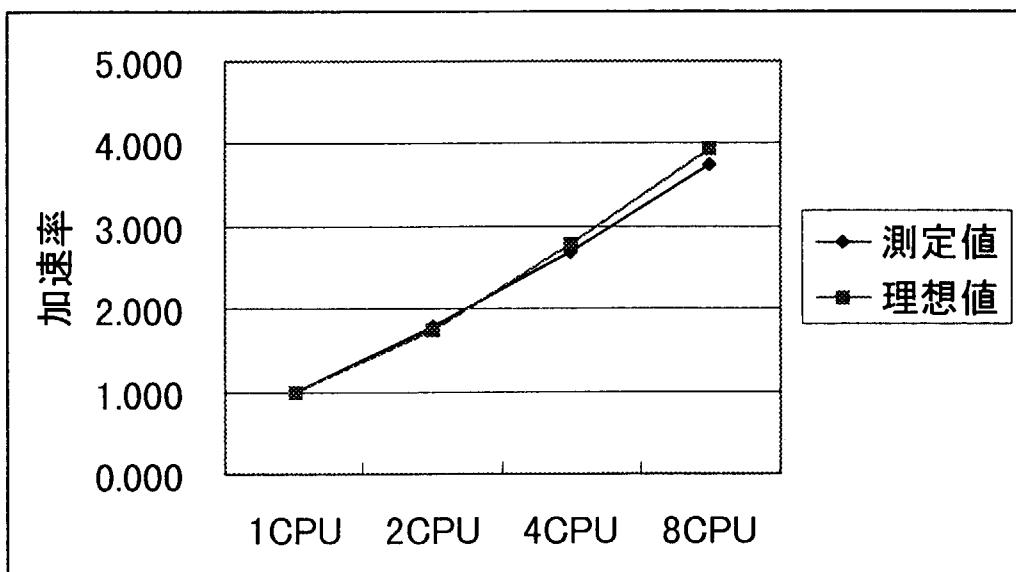


図 8 加速率(測定値と理想値)

加速率の理想値は、次のアムダールの公式から計算している。

$$\text{理想的な加速率} = 1 / \{(1 - T_{par}) + T_{par} / \text{CPU 数}\}$$

$$\begin{aligned} T_{par} &= \text{並列化可能部分の CPU 時間} / \text{逐次実行の CPU 時間} \\ &= (125.82 + 56.40) / 213.54 = 0.8533 \end{aligned}$$

上記の結果をみると、理想的な加速率が得られている。特に並列化ルーチンの全経過時間を 1CPU と 2CPU の場合で比較すると、ほぼ 2 倍の性能向上が観測されている(表5参照)。msd の

値が2倍を超えてるのは計測誤差によるものと考えられる。この結果から、SX-4 マイクロタスクによる並列化では通信オーバヘッドが極小に抑えられていることが確認される。

***** Program Information *****		
Real Time (sec)	:	57.114075
User Time (sec)	:	299.528507
Sys Time (sec)	:	0.448491
Vector Time (sec)	:	220.188702
Inst. Count	:	6880612543.
V. Inst. Count	:	2403358319.
V. Element Count	:	408790654756.
FLOP Count	:	171330220793.
MOPS	:	1379.728137
MFLOPS	:	571.999716
MOPS (concurrent)	:	7247.884753
MFLOPS (concurrent)	:	3004.786164
VLEN	:	170.091431
V. Op. Ratio (%)	:	98.916622
Memory Size (MB)	:	108.000000
Max Concurrent Proc.	:	8.
Conc. Time( $\geq$ 1) (sec)	:	57.019106
Conc. Time( $\geq$ 2) (sec)	:	34.801296
Conc. Time( $\geq$ 3) (sec)	:	34.778392
Conc. Time( $\geq$ 4) (sec)	:	34.761453
Conc. Time( $\geq$ 5) (sec)	:	34.743241
Conc. Time( $\geq$ 6) (sec)	:	34.728998
Conc. Time( $\geq$ 7) (sec)	:	34.655623
Conc. Time( $\geq$ 8) (sec)	:	34.447573
Event Busy Count	:	0.
Event Wait (sec)	:	0.000000
Lock Busy Count	:	0.
Lock Wait (sec)	:	0.000000
Barrier Busy Count	:	0.
Barrier Wait (sec)	:	0.000000
MIPS	:	22.971478
MIPS (concurrent)	:	120.672052
I-Cache (sec)	:	1.327661
O-Cache (sec)	:	22.670487
Bank (sec)	:	18.387003
real	57.74	
user	5:00.04	
sys	0.49	

図 9 プログラム性能情報(8CPU)

次にロードバランスについて調べてみよう。並列化性能の向上を妨げる原因としては、並列化通信時間の他に、ロードバランスが均等でないことがあげられる。8CPU 実行時のプログラム性能情報を見ると(図 9 参照)、並列化 revpb ではロードバランスが均等であることが確認される。それは、"Conc. Time( $\geq$  2)(sec)" から "Conc. Time( $\geq$  8)(sec)" までがほぼ同じ値を示していること

からわかる。例えば”Conc. Time( $\geq 8$ )(sec)”が 34.45 秒とは、8CPU が同時に動作した時間が 34.45 秒であったことを意味する。”Conc. Time( $\geq 1$ )(sec)”だけが他より大きな値になるのは、メインプログラム等の逐次実行部分が含まれるためである。

今回の作業では理想的な加速率は得られなかつたが、8CPU でほぼ4倍高速化を達成しており、投入した作業量を考えると満足すべき結果といえる。

この章の最後に自動並列化コンパイラについて記述しておく。既に述べたが、共有メモリ型の並列化の大きな長所として、自動並列化コンパイラのサポートがあげられる。これはコンパイラが自動的に DO ループの並列性を検知し、並列化オブジェクトを生成するものである。SX-4 でも自動並列化コンパイラがサポートされており、コンパイル時に自動並列化を指示するオプションを指定することにより利用可能である。この機能を用いれば、ユーザは並列化の概念や指示行を知らなくても、SX-4 のマイクロタスクによる並列化機能を利用することができ、ユーザの負担軽減に有効である。しかし、自動並列化が可能な DO ループは単純な多重 DO ループのみであり、たとえ多重 DO ループでもループ中で他のルーチンを call している場合には並列化されない。今回並列化した revpb では並列化対象の DO ループ中には、FFT ルーチン等の多数ルーチンが call されており、自動並列化は当初から断念せざるを得なかつた。

## 5 実空間分子動力学法プログラムの並列化

### 5.1 プログラムの性能解析

第5章と第6章では、実空間分子動力学法プログラム realmesh の並列化について述べる。第2章で述べたように、実空間分子動力学法プログラムは、ベクトル化や並列化による超高速計算を意図して開発されている。特に実空間を複数の領域に分割し、それぞれの領域で独立に分子動力学法プログラムを実行する領域分割法は、将来的には非常に有望である。それは CPU 数に比例した高速化が期待でき、しかも分散メモリ型並列計算機の大容量メモリが利用できるので、大規模な分子動力学計算が可能になるからである。しかしプログラミングという観点から見ると、領域分割法は分散メモリ型並列化手法を必要とし、プログラム全体に渡る詳細な調査と大幅な修正を必要とする。そのため本稿では領域分割法を採用せず、CPU 消費時間の大きいルーチンにある DO ループの並列化を中心に高速化を図ることにする。なお、realmesh のアルゴリズムの詳細は参考文献[7]及び付録B を参照されたい。

第3章の場合と同じく、SX-4 のプログラム性能解析機能を用いてプログラム全体の性能解析を行ったところ、次のような結果が得られた。なお計測結果を早く得るために、反復数を制限している。

***** Program Information *****		
Real Time (sec)	:	81.724909
User Time (sec)	:	74.899586
Sys Time (sec)	:	1.619111
Vector Time (sec)	:	34.922961
Inst. Count	:	3346041724.
V. Inst. Count	:	428578783.
V. Element Count	:	62159043920.
FLOP Count	:	7159473261.
MOPS	:	868.850026
MFLOPS	:	95.587622
VLEN	:	145.035280
V. Op. Ratio (%)	:	95.516872
Memory Size (MB)	:	100.031250
MIPS	:	44.673701
I-Cache (sec)	:	4.565482
O-Cache (sec)	:	0.281994
Bank (sec)	:	2.479022
real	1:22.29	
user	1:14.90	
sys	1.65	

図 10 プログラム性能情報(逐次版)

上記の結果を見ると、ベクトル化率は 95.5%とそれほど悪くはないが、平均ベクトル長が短くフロップス値が 100MFLOPS にも満たない。この数値は SX-4 のポテンシャルを活かしているとは言い難い数値である。この原因を調査するため、KMtool90 によりルーチンごとの正味経過時間を計測した。

表 6 realmesh 各ルーチンの正味経過時間

サブルーチン名	プログラム実行時間に対する比率
vect_vosko	41.24%
lapla	34.47%
dynamic	5.82%
ham	4.29%
inipot	3.76%

上記計測結果から、交換相關ポテンシャルを計算するサブルーチン vect\_vosko をさらに調査したことろ、あるDO ループで CPU 時間の大半が消費されていることが判明した。この DO ループ中は、さらに別のサブルーチン vosko を call している。このサブルーチンはライン数こそ多いものの、DO ループも call 文も含まない単純な構造である。そのためサブルーチン vosko を vect\_vosko 中にインライン展開すれば、この DO ループがベクトル化され高速化されるものと考えられる。そこで、次のコンパイルオプションでインライン展開を指示して再コンパイルし、再度プログラム性能情報を採取した。

```
> f77 -pi *.f
```

その結果が図 11 である。オリジナルに比べ 2.4 倍の高速化がなされ、インライン展開によるベクトル化が有効に機能していることが分かる。

***** Program Information *****		
Real Time (sec)	:	34.048218
User Time (sec)	:	32.232161
Sys Time (sec)	:	0.362616
Vector Time (sec)	:	28.019492
Inst. Count	:	651683875.
V. Inst. Count	:	232295312.
V. Element Count	:	58816322242.
FLOP Count	:	7549716805.
MOPS	:	1837.782798
MFLOPS	:	234.229310
VLEN	:	253.196338
V. Op. Ratio (%)	:	99.292000
Memory Size (MB)	:	103.031250
MIPS	:	20.218436
I-Cache (sec)	:	0.205278
O-Cache (sec)	:	0.204768
Bank (sec)	:	1.597330
real	34.87	
user	32.23	
sys	0.40	

図 11 プログラム性能情報(逐次インライン展開版)

vect\_vosko 自体の全経過時間を測定したところ、オリジナルの逐次版で 37.79 秒のところが、逐次

インライン展開版で 1.71 秒と、大幅な高速化がなされていた。

次に並列化すべきルーチンを特定するため、インライン展開版 `realmesh` の全経過時間のデータを採取した(表7参照)。

表 7 インライン展開版 `realmesh` の全経過時間

サブルーチン名	プログラム実行時間に対する比率
main	100.00%
lapla	64.56%
cha2vh	56.52%
ham	16.67%
inipot	9.95%
dynamic	7.69%

この表の中で `main` はメインルーチンであり、並列化は残念ながら不可能である。次の `lapla` はラプラス演算子を数値的に計算する部分であり、電子の波動関数更新部分である `dynamic` と共に、主要な部分が多重 DO ループで構成されている。これらは独立性の高い DO ループであり、最外側 DO ループを各 CPU に並列実行させることにより高速化が可能である。しかもこれらの DO ループは、ループ外へのジャンプや他サブルーチンの呼び出しのない単純な構造をしている。そのため自動並列化が適用可能と考えられる。そこでコンパイラオプションで次の自動並列化オプションを指定し、並列化ロードモジュールを作成した。

```
> f77 -P auto *.f
```

プログラム実行方法は並列化 `revpb` の場合と同様、使用する CPU 数を環境変数で指定した後、並列化ロードモジュールを実行する。

```
> setenv F_RSVTASK 4
> a.out
```

表7の中の `cha2vh` について一言述べておく。`cha2vh` は電荷密度から静電ポテンシャルを計算する部分であり、プログラム全経過時間の約 57%を占めている。このルーチンを調査したところ、計算時間の大部分は前述のルーチン `lapla` の call に費やされていた。`lapla` に対しては自動並列化で高速化を図るため、`cha2vh` の高速化作業はあえて行うことはしなかった。

## 6 並列化 realmesh の性能測定と考察

並列化 realmesh の性能測定を行ったところ、次のような結果が得られた。なお、1CPU の測定にはインライン展開版を用いている。オリジナルの逐次版からの加速率は、下記加速率の 2.4 倍であることに注意されたい。

表 8 計測時間と加速率

	1CPU	2CPU	4CPU	8CPU
時間(秒)	32.60	23.20	21.25	20.01
加速率(測定値)	1.000	1.405	1.534	1.629
加速率(理想値)	1.000	1.546	2.125	2.616

表 9 並列化ルーチンの加速率

	1CPU	2CPU	加速率
lapla	20.53sec	11.60sec	1.77
dynamic	2.47sec	2.22sec	1.11
合計	23.00sec	13.82sec	1.66

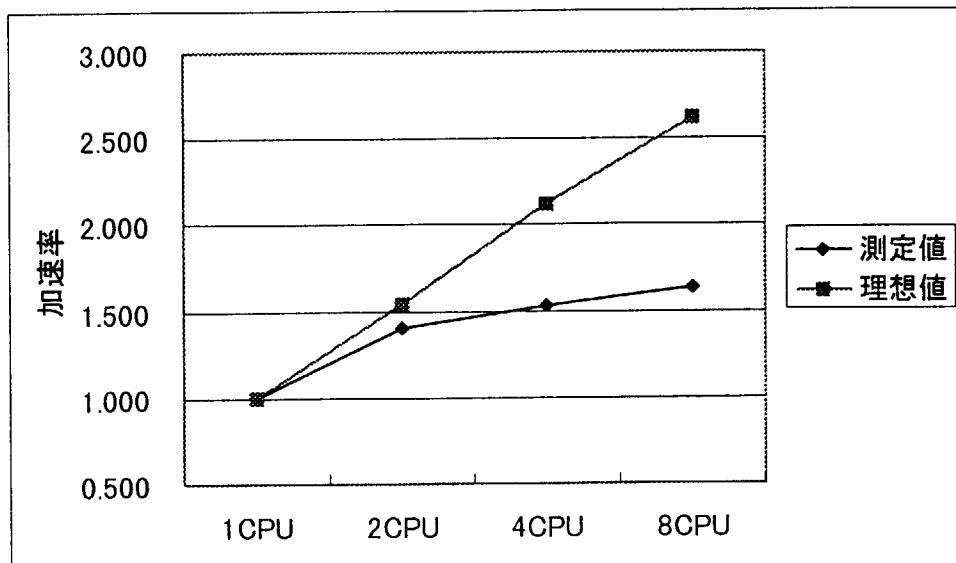


図 12 加速率(測定値と理想値)

加速率の理想値は、次のアムダールの公式から計算している。

$$\text{理想的な加速率} = 1 / \{(1 - T_{par}) + T_{par} / \text{CPU 数}\}$$

$$\begin{aligned} T_{par} &= \text{並列化可能部分の CPU 時間} / \text{逐次実行の CPU 時間} \\ &= (20.53 + 2.47) / 32.60 = 0.706 \end{aligned}$$

上記の結果をみると、CPU 数が多くなるにつれ、理想的な加速率からかい離する傾向が観察さ

れる。また並列化ルーチンの 2CPU で実行させた場合の計測時間においても、lapla は 1.77 倍とある程度の性能は出ているが、dynamic はほとんど性能向上が見られない。原因としては、CPU 数が多くなるにつれ、並列化粒度が小さくなり並列化オーバヘッドが相対的に大きくなることと、コンパイラが自動並列化を行うために余分なコードを挿入しているためと考えられる。

ロードバランスに関しては次の 8CPU 実行時のプログラム性能情報からわかるように、均等なロードバランスが実現されている。

***** Program Information *****		
Real Time (sec)	:	20.011928
User Time (sec)	:	87.072501
Sys Time (sec)	:	2.048858
Vector Time (sec)	:	33.636326
Inst. Count	:	950611630.
V. Inst. Count	:	241059446.
V. Element Count	:	61046431206.
FLOP Count	:	7888739440.
MOPS	:	709.247842
MFLOPS	:	90.599665
MOPS (concurrent)	:	3358.213064
MFLOPS (concurrent)	:	428.979775
VLEN	:	253.242228
V. Op. Ratio (%)	:	98.851039
Memory Size (MB)	:	132.000000
Max Concurrent Proc.	:	8.
Conc. Time(>= 1) (sec)	:	18.389537
Conc. Time(>= 2) (sec)	:	10.063542
Conc. Time(>= 3) (sec)	:	10.034871
Conc. Time(>= 4) (sec)	:	10.007227
Conc. Time(>= 5) (sec)	:	9.979047
Conc. Time(>= 6) (sec)	:	9.955910
Conc. Time(>= 7) (sec)	:	9.933451
Conc. Time(>= 8) (sec)	:	8.708990
Event Busy Count	:	0.
Event Wait (sec)	:	0.000000
Lock Busy Count	:	0.
Lock Wait (sec)	:	0.000000
Barrier Busy Count	:	0.
Barrier Wait (sec)	:	0.000000
MIPS	:	10.917472
MIPS (concurrent)	:	51.693070
I-Cache (sec)	:	0.395976
O-Cache (sec)	:	0.502499
Bank (sec)	:	1.681578
real	20.06	
user	1:27.08	
sys	2.08	

図 13 プログラム性能情報(8CPU 版)

## 7 おわりに

本稿では、第一原理分子動力学法プログラムと実空間分子動力学法プログラムの SX-4 共有メモリ型並列化について説明した。第一原理分子動力学法プログラムは、8CPU で 3.7 倍の高速化を達成している。これはアムダールの公式から見てもほぼ理想的な値である。一方、実空間分子動力学法プログラムでは、ベクトル化による高速化で 2.4 倍、さらに並列化により 8CPU で 1.6 倍、あわせて 3.9 倍の高速化が図られている。並列化による高速化はやや不本意な結果であり、今後原因を究明し、さらに性能向上を目指したいと考えている。

今回の並列化には SX-4 のマイクロタスク並列化機能を用いた。この機能は NEC SX シリーズ固有の機能であり他機種では利用できないこと、共有メモリ型並列化機能なので 1 ノード内の並列化にのみ有効なこと等から、一般ユーザに浸透しているとは必ずしもいえない。しかし並列化に要する作業量は、MPI のそれに比べはるかに少なく、しかも並列化による性能向上が達成しやすいという利点を有する。本稿で説明した 2 本のプログラムの並列化でも、このことは確認されたといえる。

今後の課題としては、実空間分子動力学法プログラムの並列化性能向上と共に、領域分割法による並列化実空間分子動力学法プログラムの開発があげられる。領域分割法による並列化に成功すれば、現時点の解析規模をはるかに上回る系が解析可能となり、しかも超高速でスケーラブルな計算性能が期待できる。従ってその研究価値は非常に高いといえる。しかし一方、並列化には MPI の利用が不可欠であり、プログラミングは困難を極めるものと考えられる。地道で継続的な研究開発が必要であろう。

## 謝 辞

本論文における性能測定の一部は、日本電気株式会社府中事業場の SX-4 を利用させて頂きました。日本電気株式会社スーパーコンピュータ販売推進本部の秋葉幸範主任と長沢富人氏には SX-4 利用に際し、多くのご支援を給わりました。日本原子力研究所計算科学技術推進センターの相川裕史次長には、本共同研究遂行にあたり多くのご支援を給わりました。ここに感謝の意を表します。

## 参考文献

- [1] 長嶋雲兵 監修, “計算化学”, 計算工学, Vol.4, No.4, 1999
- [2] T. OGUCHI and T. SASAKI, “Density-Functional Molecular-Dynamics Method”, Progress of Theoretical Physics Supplement No.103, 93(1991)
- [3] A. Becke, “Density-functional thermochemistry.3”, Journal of Chemical Physics, Vol.98, No.7(1993)
- [4] 渡部弘 長尾佐市 滝川好夫 熊倉利昌, “並列／非並列共用プログラム性能解析ツールの開発”, JAERI-Data/Code 99-014
- [5] 日本電気編, “FORTRAN90/SX 並列処理機能利用の手引き”, G1AF08-1
- [6] 小林一昭, <http://www.nirim.go.jp/~kobayak/INFO/guidej.html>
- [7] 新井正男, <http://www.nirim.go.jp/~arai/>
- [8] P. Hohenberg and W. Kohn, Phys. Rev. {136}, B864 (1964)
- [9] W. Kohn and L. J. Sham, Phys. Rev. {140}, A1133 (1965)
- [10] A. Nobile and V. Roberto, "MFFT: A PACKAGE FOR TWO AND THREE-DIMENSIONAL VECTORIZED DISCRETE FOURIER TRANSFORMS", Computer Physics Communications 42, p233-247(1986), North-Holland, Amsterdam
- [11] A. Nobile and V. Roberto, "EFFICIENT IMPLEMENTATION OF MULTIDIMENSIONAL FAST FOURIER TRANSFORMS ON A CRAY X-MP", Computer Physics Communications 40, p189-201(1986), North-Holland, Amsterdam

## 付 錄 A

## 第一原理分子動力学法プログラム revpb の構造

```

+-main-----TIME-----CLOCKM 時間 計測用サブルーチン (なくてもよい)
|--SIMP      シンプソン積分用係数計算
|--INPUT1    座標データ入力用
|--INFOUT   計算情報出力 (なくてもよい)
|--LATTIC----HPSORT 基本逆格子ベクトル計算 (LATTIC)
|--GSTEP1    逆格子座標計算 (2Gmax以下)
|--HPSORT    逆格子座標を小さい順に並べる
|--SYMM-----OPGR----OPMTRX 対称性計算 (SYMM)
|--GSTEPF   FFT用マッピング
|--KSTEP-----KPMSE 对称性なし (表面用) サンプリングK点計算
|   |--KPMWBZ 对称性なし (バルク用)
|   |--SCCM    単純立方用
|   |--BCCM    体心立方用
|   |--FCCM    面心立方用
|   |--HEXM    六方最密充填用
|   |--TETRAH  正方晶 (ルチル構造) 用
|   |--APBO2   斜方晶 ( $\alpha$ PbO2構造) 用
|--BASNUM   平面波用逆格子計算 (Gmax以下)
|--LATSCA   LATTICストレス用 (定圧)
|--GSTSCA   GSTEP1ストレス用 (定圧)
|--SYMSCA   SYMMストレス用 (定圧)
|--GSFSCA   GSTEPFストレス用 (定圧)
|--FORM     フォームファクター計算用
|--PCC-----DSJNV 部分内殻補正用 (PCC)
|--PSEUDO---DSJNV 局所擬ポテンシャル用 (入力、フーリエ変換)
|--PSELMD   PSEUDO (非MDループ用)
|--EWVEC    EWVEC (非MDループ用)
|--EWVMD   EWVMD (非MDループ用)
|--INTCHG   初期電荷密度設定用
|--EVIN     パラメータ入力 (なくてもよい)
|--XCFFT----C3FFT <---- MFFT 交換相関項計算用 (XCFFT)
|--KBMAT----DSJNV 非局所擬ポテンシャル部分 (入力、フーリエ変換)
|--DIAGON---CHOBSD 初期波動関数 (固有ベクトル) 設定 [対角化]
|--KBINT----DSJNV 非局所擬ポテンシャル (非初期用、フーリエ変換)
|--MSD-----KBINT--DSJNV カー・パリネロ本体 (MSD)
|   |--FORZFB  非局所部分計算用 (非MDループ用)
|   |--C3FFT  <---- MFFT 高速フーリエ変換
|   |--TIME----CLOCKM 時間計測 (なくてもよい)
|--FERMI----WIDTH2 フエルミエネルギー計算用 (FERMI)
|--FORCE    力の計算 (非局所部分) 用
|--FORZFB   非局所部分計算用 (非MDループ用)
|--CHAVER---C3FFT <---- MFFT 電荷密度計算用 (CHAVER)
|   |--CHGAVR  電荷密度対称性計算用
|--STRESS---XCFFT----C3FFT <---- MFFT ストレス計算用 (STRESS)
|   |--XSTPC---C3FFT <---- MFFT (局所部分用)
|   |--STRNL   ストレス計算 (非局所部分用)
|--FORLOC   力の計算用 (局所部分)
|--FORCES   力の対称性計算用
|--MD       古典的分子動力学部分用
|--ENERGY---XCFFT----C3FFT <---- MFFT 全エネルギー計算用 (ENERGY)
|--CONV2    収束判定用
|--EVOUT    計算結果、パラメーター出力用 (その1)
|--EVOUT2   計算結果、パラメーター出力用 (その2)

```

## 並列化ソースコード

```

subroutine chaver_sub
&   (CHGB1_, KV3_, NBD1_, NBD2_, IBA_, KSUM_, KFT1_, KFT2_, KFT3_,
&   IWL, IWM, IWN)
IMPLICIT REAL (A-H, O-Y)
IMPLICIT COMPLEX (Z)
INCLUDE 'PACVPP'
dimension CHGB1_(*), IBA_(*), CHGB1_local (KFFT)

```

```

DIMENSION ZC3D1(IFX2,IFY2,IFZ2)
DIMENSION IWL(*), IWM(*), IWN(*), IWORK(2*IFX2)
EQUIVALENCE (ZC1D(1),ZC3D1(1,1,1))

!
do i = 1, KSUM_
    CHGB1_local(i) = 0. do
end do
!
*pdifr pardo
DO 7820 IK=1,KV3_
    DO 7810 IBAN=NBD1_, NBD2_
C***** BUILD UP CHARGE DENSITY FOR IBAN-TH BAND, IK-TH K-POINT
    IIBA=IBA_(IK)
    DO 7830 I=1, KSUM_
        ZC1D(I)=DCMPLX(0.0D0, 0.0D0)
7830    CONTINUE
    DO 232 I=1, IIBA
        I1=NBASE(I, IK)
        L1=IGF1(I1)
        L2=IGF2(I1)
        L3=IGF3(I1)
cGEN      ZC3D1(L1,L2,L3) = ZZZ(I, IBAN, IK)
        ZC3D1(L1,L2,L3) = ZAJ(I, IBAN, IK)
232      CONTINUE
C***** INVERSE FAST FOURIER TRANSFORMATION ----
    CALL C3FFT(ZC3D1, KFT1_, KFT1_-1, KFT2_, KFT3_, IWL, IWM, IWN
    & , 0, 1, 1, IWORK, IERR1)
    IF (IERR1.NE. 0) THEN
        WRITE (6,*)' C3FFT(IFFT C)] IERR1 = ', IERR1
cGEN      STOP
    END IF
    DO 240 I=1, KSUM_
        CHGB1_local(I)=CHGB1_local(I)+OCCUP(IBAN, IK)*
    & DCONJG(ZC1D(I))*ZC1D(I)
240      CONTINUE
7810    CONTINUE
7820    CONTINUE
!
*pdifr critical
    do i=1, KSUM_
        CHGB1_(i) = CHGB1_(i) + CHGB1_local(i)
    end do
*pdifr end critical
!
    return
end

SUBROUTINE MSD(IREC8,IREC9,IMD,ICAR,ISTR)
C~~~~~  

C
    IMPLICIT REAL (A-H, O-Y)
    IMPLICIT COMPLEX(Z)
    INCLUDE 'PACVPP'
C VPP-PARALLEL
!XOCL SUBPROCESSOR PS(IPARA)=PQ(1:IPARA)
!XOCL INDEX PARTITION IP=(PS, INDEX=1:KNV3, PART=BAND)
    DIMENSION SSS(KNG1,KNV3,KTYP,10), ZZZ(KNG1,KEG,KNV3)
    & , ZFC(KEG,KNV3,KATM,10)
    DIMENSION EKK(KEG,KNV3), OCCUU(KEG,KNV3), RAA(KNG1,KEG,KNV3)
!XOCL LOCAL SSS(:, /IP, :, :), ZZZ(:, :, /IP), ZFC(:, /IP, :, : )
!XOCL LOCAL EKK(:, /IP), OCCUU(:, /IP), RAA(:, /IP)
    EQUIVALENCE (SNL,SSS), (ZAJ,ZZZ), (ZFBB,ZFC)
    & , (EKO,EKK), (OCCUP,OCCUU), (RAK,RAA)
C EIGEN-VALUE PROBLEM
    DIMENSION ZDEVK(KNG1,KEG)
    & , EMAS(KEG)
C-----ARRAYS FOR MFFT-----

```

```

      DIMENSION ZEV3C(IFX2,IFY2,IFZ2),ZV3D(IFX2,IFY2,IFZ2)
      DIMENSION IWL(8*IFX2+28),IWM(8*IFY2+28),IWN(8*IFZ2+28)
      & ,IWORK(2*IFX2)
C=====
      EQUIVALENCE (ZV1D(1),ZV3D(1,1,1)),(ZC1D(1),ZEV3C(1,1,1))
cGEN
      DIMENSION CWLNEW(10)
cGEN
      ATTN KX1 OR KX11 =====
C REWIND 90
      KFT1 = IFX2
      KFT2 = IFY2
      KFT3 = IFZ2
      KSUM=KFT1*KFT2*KFT3
      KVOL=(KFT1-1)*KFT2*KFT3
cGEN
*pdif serial
      IF (ITER.EQ.2) WRITE (6,*) 'DTIME = ',DTIM
C----- MASS OF ELECTRON -----
      DO 4 I=1,KG
         ZCHG0(I) = ZCHG(I)
         ZCHG(I) = DCMLX(0.0D0,0.0D0)
      4 CONTINUE
C     FOR STRESS (CALL KBINT & FORZFB 1992 1/7)
      IF(ISTR.EQ.1) THEN
         WRITE (6,*) 'KBINT ITER>IMD IN MSD'
C
      CALL KBINT
C----- STRESS
      CALL FORZFB
C
      END IF
C /////////////////
C // CALCULATE THE VERLET ALGORITHM //
C /////////////////
      ZVP(1) = ZVXC(1) + ZPSCC(1)
      DO 2001 I=2,KG
         ZVP(I) = ZVXC(I) + ZPSCC(I) + PA14*ZCHG0(I)*RGG(I)
      2001 CONTINUE
cGEN
*pdif endserial
C***** SET UP C3FFT (FAST FOURIER TRANSFORMATION) -----
      CALL C3FFT(ZEV3C,KFT1,KFT1-1,KFT2,KFT3,IWL,IWM,IWN
      & ,0,0,1,IWORK,IERR)
      IF (IERR.NE.0) THEN
         WRITE (6,*) ' C3FFT(SET UP)] IERR = ',IERR
         STOP
      END IF
      DO 234 I=1,KSUM
         ZV1D(I)=DCMLX(0.0D0,0.0D0)
      234 CONTINUE
      DO 233 I=1,KG
         LF1=IGF1(I)
         LF2=IGF2(I)
         LF3=IGF3(I)
         ZV3D(LF1,LF2,LF3) = ZVP(I)
      233 CONTINUE
cGEN CALL TTTT(ITIME1,'      ',1)
C***** INVERSE FAST FOURIER TRANSFORMATION -----
      CALL C3FFT(ZV3D,KFT1,KFT1-1,KFT2,KFT3,IWL,IWM,IWN
      & ,0,1,1,IWORK,IERR2)
cGEN CALL TTTT(ITIME1,'C3FFT ',2)
      IF (IERR2.NE.0) THEN
         WRITE (6,*) ' C3FFT(IFTT V)] IERR2 = ',IERR2
         STOP
      END IF

```

```

C      WRITE(6,*) 'ZV1D = '
C      DO 27 I=1,20
C         WRITE(6,*) ZV1D(I)
C 27 CONTINUE
C-----K-POINT LOOP-----
C      VPP-PARALLEL START 2
C!XOCL PARALLEL REGION
!XOCL SPREAD DO /IP
cGEN
*pdir pardo
      DO 100 NNN=1, KV3
C
      IWRT(NNN) =NNN
      AKX = VX(NNN)
      AKY = VY(NNN)
      AKZ = VZ(NNN)
      IIIBA = IIIBA(NNN)

      DO 3 IBAN=NBD1, NBD2
      EMAS(1BAN)=0.0D0
3 CONTINUE
      DO 2002 J=1, IIIBA
      X(J) = 0.0D0
      VKIN =
      &          ( (AKX+GX(NBASE(J, NNN)))**2
      &          + (AKY+GY(NBASE(J, NNN)))**2
      &          + (AKZ+GZ(NBASE(J, NNN)))**2 )/2. DO
      Y3(J) = VKIN + DREAL(ZVP(1))
2002 CONTINUE
      DO 7000 ITY=1, KTYP
      CS=1.0D0/(WS(ITY)*UNIVOL)
      CP=1.0D0/(WP(ITY)*UNIVOL)
cGEN  CWL(1)=CS
cGEN  CWL(2)=CP
cGEN  CWL(3)=CP
cGEN  CWL(4)=CP
      CWLNEW(1)=CS
      CWLNEW(2)=CP
      CWLNEW(3)=CP
      CWLNEW(4)=CP
      IF (NLSPD(ITY).EQ.1) THEN
      LNUM = 4
      ELSE
      LNUM = 9
      CD=1.0D0/(WD(ITY)*UNIVOL)
cGEN  CWL(5)=CD
cGEN  CWL(6)=CD
cGEN  CWL(7)=CD
cGEN  CWL(8)=CD
cGEN  CWL(9)=CD
      CWLNEW(5)=CD
      CWLNEW(6)=CD
      CWLNEW(7)=CD
      CWLNEW(8)=CD
      CWLNEW(9)=CD
      END IF
C
      DO 7111 L=1, LNUM
      DO 7010 I=1, IIIBA
cGEN   TMPP      = CWL(L)*SSS(I, NNN, ITY, L)**2
cGEN   TMPP      = CWLNEW(L)*SSS(I, NNN, ITY, L)**2
      Y3(I) = Y3(I) + TMPP*DFLOAT(IATOM(ITY))
      X(I) = X(I) + TMPP*DFLOAT(IATOM(ITY))
7010  CONTINUE
7111  CONTINUE
7000 CONTINUE
C-----BAND LOOP-----
      DO 200 IBAN=NBD1, NBD2

```

```

C*****REC = KV3*(IBAN-1)+WRT(NNN)
C      IF (IREC8.NE.0) IREC=IREC8*IREC-IREC8+1
C      READ(80, REC=IREC) ZV1
C      IREC = KV3*(IBAN-1)+WRT(NNN)
C      IF (IREC.LE.1000) THEN
C          IF (IREC.LE.IRECMX) THEN
C              IF (IREC9.NE.0) IREC=IREC9*IREC-IREC9+1
C              READ(90, REC=IREC) ZRC
C          ELSE
C              READ(90, REC=IREC) ZRC
C-----90. 1. 22 Y. M. -----
C-----90. 1. 22 Y. M. -----
DO 3231 I=1,KSUM
ZC1D(I)=CMPLX(0.000,0.000)
3231 CONTINUE
DO 3232 I=1,IIBA
I1=NBASE(I,NNN)
L1=IGF1(I1)
L2=IGF2(I1)
L3=IGF3(I1)
ZEV3C(L1,L2,L3) = ZZZ(I,IBAN,NNN)
C*****ZEV3C(L1,L2,L3) = ZV1(I)
3232 CONTINUE
C*****---- INVERSE FAST FOURIER TRANSFORMATION ----
CALL C3FFT(ZEV3C,KFT1,KFT1-1,KFT2,KFT3,IWL,IMM,INW
& ,0,1,1,IWORK,IERR1)
IF (IERR1.NE.0) THEN
    WRITE (6,*) ' C3FFT(IFFT C)] IERR1 = ', IERR1
CGEN STOP
END IF
CXX DO 3240 I=1,KSUM
CXX ZRC(I)=ZC1D(I)
C3240 CONTINUE
C      END IF
C      ----- CALCULATE THE MATRIX ELEMENTS -----
DO 71 I=1,KNG1
C      ZPNL(I) = - ZZZ(I,IBAN,NNN)*X(I)
Y1(I) = - DREAL(ZZZ(I,IBAN,NNN)*X(I))
Y2(I) = - DIMAG(ZZZ(I,IBAN,NNN)*X(I))
71 CONTINUE
DO 401 IA=1,KATM
C
IF (NLSPD(KFTYPE(IA)).EQ.1) THEN
    LNUM = 4
ELSE
    LNUM = 9
END IF
C
DO 411 L =1,LNUM
DO 400 I =1,IIBA
ZTMP = SSS(I,NNN,KFTYPE(IA),L)
& * ZFC(IBAN,NNN,IA,L)
C      ZPNL(I) = ZPNL(I) + ZTMP*ZFM2(NBASE(I,NNN),IA)
Y1(I) = Y1(I) + DREAL(ZTMP*ZFM2(NBASE(I,NNN),IA))
Y2(I) = Y2(I) + DIMAG(ZTMP*ZFM2(NBASE(I,NNN),IA))
400 CONTINUE
411 CONTINUE
401 CONTINUE
DO 240 I=1,KSUM
C      ZC1D(I)=ZV1D(I)*ZRC(I)
ZC1D(I)=ZV1D(I)*ZC1D(I)
240 CONTINUE
C*****---- FAST FOURIER TRANSFORMATION ----
CALL C3FFT(ZEV3C,KFT1,KFT1-1,KFT2,KFT3,IWL,IMM,INW
& ,0,-1,1,IWORK,IERR)
IF (IERR.NE.0) THEN
    WRITE (6,*) ' C3FFT(FFT)] IERR = ', IERR

```

```

cGEN      STOP
END IF
DENOM=1. ODO/DBLE (KVOL)
DO 30 I=1, 11BA
  I1=NBASE(I, NNN)
  L1=IGF1(I1)
  L2=IGF2(I1)
  L3=IGF3(I1)
  ZPGG=ZEV3C(L1, L2, L3)*DENOM
  WDI =(Y3(I)-EKK(1BAN, NNN))
C   ZROF =(ZPGG+ZPNL(I)-ZVP(1)*ZZZ(I, 1BAN, NNN))
  ZROF =(ZPGG+Y1(I)+Z1*Y2(I)-ZVP(1)*ZZZ(I, 1BAN, NNN))
  ZDEVC(I, 1BAN)=-(WDI)*ZZZ(I, 1BAN, NNN)+ZROF
  EMAS(1BAN)=EMAS(1BAN)+DCONJG(ZZZ(I, 1BAN, NNN))*ZDEVC(I, 1BAN)
  ESHI=DEXP(-WDI*DTIME)
  ZZZ(I, 1BAN, NNN)=ZZZ(I, 1BAN, NNN)*ESHI+(ESHI-1. ODO)*ZROF/WDI
30 CONTINUE
C-----
C 200 CONTINUE
C >>> AFTER C(T+DT)=C(T)+DC(T) <<<
C~~~~~
C GRAM-SCHMIDT ORTHOGONALIZATION
C~~~~~
C   IF (NNN.EQ.1) CALL TTTT(ITIME1,      ', 1)
C   CALL GRAMMD(NNN, 11BA)
C   IF (NNN.EQ.1) CALL TTTT(ITIME1, 'GRAM ', 2)
DO 1000 IBAN=NBD1, NBD2-1
  ZNRM=DCMPLX(0. ODO, 0. ODO)
  DO 1010 I=1, 11BA
    ZNRM=ZNRM+DCONJG(ZZZ(I, 1BAN, NNN))*ZZZ(I, 1BAN, NNN)
1010 CONTINUE
  DENOM=1. ODO/DSQRT(DREAL(ZNRM))
  DO 1020 I=1, 11BA
    ZZZ(I, 1BAN, NNN)=ZZZ(I, 1BAN, NNN)*DENOM
1020 CONTINUE
  DO 1030 JBAN=1BAN+1, NBD2
    ZNRM=DCMPLX(0. ODO, 0. ODO)
    DO 1040 I=1, 11BA
      ZNRM=ZNRM+DCONJG(ZZZ(I, 1BAN, NNN))*ZZZ(I, JBAN, NNN)
1040 CONTINUE
  DO 1050 I=1, 11BA
    ZZZ(I, JBAN, NNN)=ZZZ(I, JBAN, NNN)-ZNRM*ZZZ(I, 1BAN, NNN)
1050 CONTINUE
1030 CONTINUE
1000 CONTINUE
  ZNRM=DCMPLX(0. ODO, 0. ODO)
  DO 1100 I=1, 11BA
    ZNRM=ZNRM+DCONJG(ZZZ(I, NBD2, NNN))*ZZZ(I, NBD2, NNN)
1100 CONTINUE
  DENOM=1. ODO/DSQRT(DREAL(ZNRM))
  DO 1110 I=1, 11BA
    ZZZ(I, NBD2, NNN)=ZZZ(I, NBD2, NNN)*DENOM
1110 CONTINUE
C~~~~~
C CHECK
C*****
C   IF (ITER.GT.40) THEN
C   DO 700 1BAN=NBD1, NBD2
C     DO 720 JBAN=NBD1, 1BAN
C       ZNORM=DCMPLX(0. ODO, 0. ODO)
C       DO 7001 I=1, 11BA
C         ZNORM=ZNORM+DCONJG(ZAJ(I, 1BAN, NNN))*ZAJ(I, JBAN, NNN)
C7001 CONTINUE
C     RNOR=DSQRT(CDABS(ZNORM))
C     IF (1BAN.EQ.JBAN) THEN
C       IF (DABS(1. ODO-RNOR).GT. 1. OD-7) THEN

```

```

C      WRITE(6,*) 'NRM, NE, 1, 0 I= ', IBAN, 'NRM= ', RNOR
C      END IF
C      ELSE
C          IF(RNOR.GT.1.0D-7) THEN
C              WRITE(6,*) 'NRM, NE, 0, 0 I J= ', IBAN, JBAN
C          &           , 'NRM= ', RNOR
C          END IF
C          IFLAG=1
C      END IF
C 720  CONTINUE
C 700  CONTINUE
C  END IF
C*****+
DO 261 IBAN=NBD1,NBD2
DO 260 I=1,1IBA
EKK(IBAN,NNN)=EKK(IBAN,NNN)
&           -2.0D0*DCONJG(ZZZ(I,IBAN,NNN))*ZDEV(I,IBAN)
260 CONTINUE
EKK(IBAN,NNN)=EKK(IBAN,NNN)+EMAS(IBAN)
C     EG(IBAN)=EKK(IBAN,NNN)
261 CONTINUE
DO 262 IBAN=NBD1,NBD2-1
DO 262 JBAN=IBAN+1,NBD2
IF(EKK(JBAN,NNN).LT.EKK(IBAN,NNN)) THEN
EE =EKK(IBAN,NNN)
EKK(IBAN,NNN)=EKK(JBAN,NNN)
EKK(JBAN,NNN)=EE
DO 270 IV=1,1IBA
ZTV           = ZZZ(IV,IBAN,NNN)
ZZZ(IV,IBAN,NNN) = ZZZ(IV,JBAN,NNN)
ZZZ(IV,JBAN,NNN) = ZTV
270 CONTINUE
END IF
262 CONTINUE
C     DO 263 IBAN=NBD1,NBD2
C     EKK(IBAN,NNN)=EG(IBAN)
C     WRITE(6,*) 'NNN = ', NNN, 'IBAN = ', IBAN
C     WRITE(6,*) EKO(IBAN,NNN)
C 263 CONTINUE
C     OUTPUT EIGENVECTORS ON FILE 80 (DIRECT-ACCESS) (KG1 <--> 1IBA)
DO 231 IBAN = NBD1,NBD2
C*****!REC = KV3*(IBAN-1)+IWRT(NNN)
C           IF (IREC8.NE.0) IREC=IREC8*IREC-IREC8+1
C*****WRITE(80,REC=IREC) ZV1
231 CONTINUE
100 CONTINUE
!XOCL END SPREAD
C!XOCL END PARALLEL
RETURN
END

```

## 付 錄 B

## 実空間分子動力学法プログラム realmesh の構造

```

+-main----INIMES          # 実空間格子点を生成
|  |--INIPSU---SPLINE    # 各原子擬ポテンシャルを読みこむ
|  |  |--SPLINT
|  |  |--ABIBUN
|  |--INIPOT---INTPOL    # 実空間格子点上のポテンシャルを初期化
|  |--INIVCORE           # 各原子のコア(内殻)ポテンシャルを初期化
|  |--INIMAD              # マーデルングポテンシャルを初期化
|  |--INIMA2
|  |--WFINIT              # 波動関数を初期化
|  |--NEWCHA               # 電荷密度を計算
|  |--SYMCHA               # 電荷密度を対称化
|  |--CHA2VH-|
|  |  |----LAPLA          # ラプラス演算子
|  |  |--PC_HAR            # 前処理
|  |
|  |--NEWPOT-|             # ポテンシャルの更新
|  |  |---VECT_V---VOSKO  # 交換相関ポテンシャルの計算
|  |
|  |--HAM-----LAPLA-----SETSAB # シュレディンガー方程式の計算
|  |  |--NONLOC
|  |--ETOTCA               # 全エネルギーの計算
|  |
|  |--DYNAMI--             # 波動関数の更新
|  |  |---PRECON           # 前処理
|  |
|  |--REORTH               # 波動関数再直交化
|  |--ATMOVE               # 原子を動かす
|  |--CALCOC               # 軌道の占有率
|  |--MIXCHA               # 電荷密度を更新
|  |--AFORCE---NLFORC      # 原子に働く力を更新
|  |
|  |--DIAGO                 # 対角化によりエネルギーレベルを計算

```

## 並列化対象部分ソースコード

```

*****
*      Laplacian Operator.
*      IPBC = 1 : Periodic B.C.
*      0 : Fixed B.C. (not. yet)
*****
subroutine lapla(wfin, wfout, iin)
implicit real*8 (a-h, o-z)
include 'global.f'
real*8 wfin(MPMAX), wfout(MPMAX)
integer iin
integer ixyzde(3,100)
double precision coef(100)
real*8 dsum, dgen0, dgen1
data dsum/0.0/
call etime(dgen0)

iord = iin
if ( iord .le. 0 ) then
  iord = ISABUN
endif
call setsabun(dxyzlen, ixyzde, coef, iord, nmax)

do inum = 1, MMM
  wfout(inum) = 0.0d0
endo

```

```

do idum = 1, nmax
  do inum = 1, MMAX
    ix = inum2x(inum)+ixyzde(1, idum)
    iy = inum2y(inum)+ixyzde(2, idum)
    iz = inum2z(inum)+ixyzde(3, idum)
    ixx = nperiod(1, ix)
    iyy = nperiod(2, iy)
    izz = nperiod(3, iz)
    inb = ixyz2num(ixx, iyy, izz)
    if (inb .ne. 0) then
c-----< - Delta !!!!>-----
      wfout(inum) = wfout(inum)-coef(idum)*wfin(inb)
    endif
  enddo
enddo
call etime(dgen1)
dsum = dsum + (dgen1 - dgen0)
write(*,*) ' dsum in lapla = ', dsum

return
end

*****
* before you call this routine.
* you must call ham
* s.t. wfnew = Hwfold1
* in this routine, wfold2 will be broken
* out: ilgflg = 1 if exact lagra is used
*      = 0
*      =-1
*****
subroutine dynamic(deltat, gamma, ilgflg)
implicit real*8 (a-h, o-z)
include 'global.f'

real*8 fac(3)
real*8 aaa(NMAX, NMAX), bbb(NMAX, NMAX), xmm(NMAX, NMAX)
c-----
c     real etime, tarry(2)
c     external etime
c-----
real*8 dsum, dgen0, dgen1
data dsum/0.0d0/
call etime(dgen0)

fac(1) = 2.0d0 / (1.0d0 + deltat*gamma)
fac(2) = -(1.0d0 - deltat*gamma)/(1.0d0 + deltat*gamma)
fac(3) = -deltat**2/(1+deltat*gamma)
c-----
```

ilgflg = 0

```

c-----< Preconditioning >-----
c     secst = etime(tarry)
      if (IPREC.eq.1) then
        call precond(wfnew)
      elseif (IPREC.eq.2) then
        call precond2(wfnew)
      elseif (IPREC.eq.3) then

        write(*,*) 'Not Supported using IPREC=1'
        call precond(wfnew)

      elseif (IPREC.lt.0) then
```

```

do id = 1, -IPREC
    call precondition(wfnew)
enddo

endif
c     secr = etime(tarry)
write(*,*) '      SEC:DYNAMIC:PC ', secr - secst

if ( IORDER .eq. 1 ) then
    do ist = 1, NNNST
        do inum = 1, MMAXP
            wfnew(inum, ist) = wfold1(inum, ist)
$             - deltat*wfnew(inum, ist)
            wfold2(inum, ist) = wfold1(inum, ist)
        enddo
    enddo
else
    do ist = 1, NNNST
        do inum = 1, MMAXP
            wfnew(inum, ist) =
$                 fac(1)*wfold1(inum, ist)
$                 +fac(2)*wfold2(inum, ist)
&                 +fac(3)*wfnew(inum, ist)
            wfold2(inum, ist) = wfold1(inum, ist)
        enddo
    enddo
endif
*
* Now wfnew is wf_bar
*
if ( IPREC .eq. 1 ) then
    call precondition(wfold2)
elseif ( IPREC .eq. 2 ) then
    call precondition2(wfold2)
elseif (IPREC .eq. 3 ) then
    write(*,*) 'Not Supported using IPREC=1'
    call precondition(wfold2)

elseif ( IPREC .lt. 0 ) then
    do id = 1, -IPREC
        call precondition(wfold2)
    enddo

endif
c-----< Lagrange Multiplier Term. >-----
if ( abs(ILAGRA) .eq. 1 ) then
    do 390 ist = 1, NNNST
        do 391 jst = 1, NNNST
            aaa(ist, jst) = 0.0d0
            bbb(ist, jst) = 0.0d0
            xmm(ist, jst) = 0.0d0
            do 400 inum = 1, MMAXP
                aaa(ist, jst) = aaa(ist, jst)
&                 + acell * wfnew(inum, ist)*wfnew(inum, jst)
                bbb(ist, jst) = bbb(ist, jst)
&                 + acell * wfold2(inum, ist)*wfnew(inum, jst)
                xmm(ist, jst) = xmm(ist, jst)
&                 + acell * wfold2(inum, ist)*wfold2(inum, jst)
400         continue
391         continue
390         continue
        call lagra(aaa, bbb, xmm, elagra, NNNST, NEMAX, ilgflg)

```

```

else
do ist = 1, NNNST
do jst = 1, NNNST
elagra(ist,jst) = eham(ist,jst) * deltat
enddo
enddo
endif

if ( ILAGRA .gt. 0 ) then
c
do ist = 1, NNNST
do jst = 1, NNNST
do inum = 1, MMMP
wfnew(inum,ist) = wfnew(inum,ist)
&           + wfold2(inum,jst)*elagra(jst,ist)
enddo
enddo
enddo
else
do ist = 1, NNNST
do inum = 1, MMMP
wfnew(inum,ist) = wfnew(inum,ist)
&           + wfold2(inum,ist)*elagra(ist,ist)
enddo
enddo
endif
call etime(dgen1)
dsum = dsum + (dgen1 - dgen0)
write(*,*) ' dsum in dynamic = ', dsum

return
end

```

This is a blank page.

## 国際単位系(SI)と換算表

表1 SI基本単位および補助単位

量	名称	記号
長さ	メートル	m
質量	キログラム	kg
時間	秒	s
電流	アンペア	A
熱力学温度	ケルビン	K
物質量	モル	mol
光强度	カンデラ	cd
平面角	ラジアン	rad
立体角	ステラジアン	sr

表3 固有の名称をもつSI組立単位

量	名称	記号	他のSI単位による表現
周波数	ヘルツ	Hz	s <sup>-1</sup>
力	ニュートン	N	m·kg/s <sup>2</sup>
圧力、応力	パスカル	Pa	N/m <sup>2</sup>
エネルギー、仕事、熱量	ジュール	J	N·m
功率、放射束	ワット	W	J/s
電気量、電荷	クーロン	C	A·s
電位、電圧、起電力	ボルト	V	W/A
静電容量	ファラード	F	C/V
電気抵抗	オーム	Ω	V/A
コンダクタンス	ジーメンス	S	A/V
磁束	ウェーバ	Wb	V·s
磁束密度	テスラ	T	Wb/m <sup>2</sup>
インダクタンス	ヘンリー	H	Wb/A
セルシウス温度	セルシウス度	°C	
光束度	ルーメン	lm	cd·sr
照度	ルクス	lx	lm/m <sup>2</sup>
放射能	ベクレル	Bq	s <sup>-1</sup>
吸収線量	グレイ	Gy	J/kg
線量等量	シーベルト	Sv	J/kg

表2 SIと併用される単位

名称	記号
分、時、日	min, h, d
度、分、秒	°, ', "
リットル	L, l
トン	t
電子ボルト	eV
原子質量単位	u

$$1 \text{ eV} = 1.60218 \times 10^{-19} \text{ J}$$

$$1 \text{ u} = 1.66054 \times 10^{-27} \text{ kg}$$

表5 SI接頭語

倍数	接頭語	記号
$10^{18}$	エクサ	E
$10^{15}$	ペタ	P
$10^{12}$	テラ	T
$10^9$	ギガ	G
$10^6$	メガ	M
$10^3$	キロ	k
$10^2$	ヘクト	h
$10^1$	デカ	da
$10^{-1}$	デシ	d
$10^{-2}$	センチ	c
$10^{-3}$	ミリ	m
$10^{-6}$	マイクロ	μ
$10^{-9}$	ナノ	n
$10^{-12}$	ピコ	p
$10^{-15}$	フェムト	f
$10^{-18}$	アト	a

(注)

1. 表1~5は「国際単位系」第5版、国際度量衡局1985年刊行による。ただし、1eVおよび1uの値はCODATAの1986年推奨値によった。

2. 表4には海里、ノット、アール、ヘクタールも含まれているが日常の単位なのでここでは省略した。

3. barは、JISでは流体の圧力を表わす場合に限り表2のカテゴリーに分類されている。

4. EC閣僚理事会指令ではbar, barnおよび「血圧の単位」mmHgを表2のカテゴリーに入れている。

### 換 算 表

力	N(=10 <sup>5</sup> dyn)	kgf	lbf
1	0.101972	0.224809	
9.80665	1	2.20462	
4.44822	0.453592	1	

$$\text{粘度 } 1 \text{ Pa}\cdot\text{s} = 10 \text{ P(ボアズ)(g/(cm\cdot s))}$$

$$\text{動粘度 } 1 \text{ m}^2/\text{s} = 10^4 \text{ St(ストークス)(cm}^2/\text{s)}$$

圧力	MPa(=10bar)	kgf/cm <sup>2</sup>	atm	mmHg(Torr)	lbf/in <sup>2</sup> (psi)
力	0.0980665	1	0.967841	735.559	14.2233
	0.101325	1.03323	1	760	14.6959
	$1.33322 \times 10^{-4}$	$1.35951 \times 10^{-3}$	$1.31579 \times 10^{-3}$	1	$1.93368 \times 10^{-2}$
	$6.89476 \times 10^{-3}$	$7.03070 \times 10^{-2}$	$6.80460 \times 10^{-2}$	51.7149	1

エネルギー・仕事・熱量	J(=10 <sup>7</sup> erg)	kgf·m	kW·h	cal(計量法)	Btu	ft·lbf	eV	1 cal = 4.18605J (計量法)	
								= 4.184J (熱化学)	= 4.1855J (15°C)
1	0.101972	$2.77778 \times 10^{-7}$	0.238889	$9.47813 \times 10^{-4}$	0.737562	$6.24150 \times 10^{-8}$		= 4.1868J (国際蒸気表)	
9.80665	1	$2.72407 \times 10^{-6}$	2.34270	$9.29487 \times 10^{-3}$	7.23301	$6.12082 \times 10^{-8}$			
$3.6 \times 10^6$	$3.67098 \times 10^5$	1	$8.59999 \times 10^5$	3412.13	$2.65522 \times 10^6$	$2.24694 \times 10^{-25}$			
4.18605	0.426858	$1.16279 \times 10^{-6}$	1	$3.96759 \times 10^{-3}$	3.08747	$2.61272 \times 10^{-19}$			
1055.06	107.586	$2.93072 \times 10^{-4}$	252.042	1	778.172	$6.58515 \times 10^{21}$			
1.35582	0.138255	$3.76616 \times 10^{-7}$	0.323890	$1.28506 \times 10^{-3}$	1	$8.46233 \times 10^{18}$			
$1.60218 \times 10^{19}$	$1.63377 \times 10^{20}$	$4.45050 \times 10^{26}$	$3.82743 \times 10^{20}$	$1.51857 \times 10^{-22}$	$1.18171 \times 10^{-19}$	1			

放射能	Bq	Ci	吸収線量	Gy	rad
				1	$100$
	$3.7 \times 10^{10}$	1		0.01	1

照射線量	C/kg	R	線量当量	Sv	rem
				1	100
	$2.58 \times 10^{-4}$	1		0.01	1

(86年12月26日現在)

