JAERI-Data/Code
96-023

分子動力学コードの段階的並列化手法

1996年7月

折居茂夫*・大田敏郎**

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.

編集兼発行　日本原子力研究所
印　　刷　㈱原子力資料サービス

JAERI－Data／Code 96－023

分子動力学コードの段階的並列化手法

日本原子力研究所計算科学技術推進センター
折居　茂夫*・大田　敏郎**

（1996年6月5日受理）

分子動力学法を用いた数値シミュレーションコードの並列化を，２フェーズ法と呼ばれる並列化方法を用い，段階的に行った。その結果，ベクトル並列計算機VPP500とスカラ並列計算機Paragonにおいて，ある計算パラメータの範囲では，doループのインデックスを使用して計算を各プロセッサに割り当てる並列化方法で並列性能を得られることがわかった。またVPP500では，広い範囲の計算パラメータで並列性能が得られた。この理由は，doループレベルでは並列性能が出ない計算が，ベクトル化により時間コスト的に無視できるようになったためである。また，ベクトル化のためプログラムのより狭い範囲に時間コストが集中し，その部分の並列性能が出てきたためである。本報告書は，VPP500とParagonにおける分子動力学コードの段階的並列化方法とその並列性能を示す。

日本原子力研究所：〒153　東京都目黒区中目黒２－２－54
＊　業務協力員　富士通株式会社
＊＊業務協力員　㈱CRC総合研究所

JAERI－Data／Code 96－023

Step by Step Parallel Programming Method for Molecular Dynamics Code

Shigeo ORII* and Toshio OHTA**

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

Parallel programming for a numerical simulation program of molecular dynamics is carried out with a step-by-step programming technique using the two phase method. As a result, within the range of a certain computing parameters, it is found to obtain parallel performance by using the level of parallel programming which decomposes the calculation according to indices of do-loops into each processor on the vector parallel computer VPP500 and the scalar parallel computer Paragon. It is also found that VPP500 shows parallel performance in wider range computing parameters. The reason is that the time cost of the program parts, which can not be reduced by the do-loop level of the parallel programming, can be reduced to the negligible level by the vectorization. After that, the time consuming parts of the program are concentrated on less parts that can be accelerated by the do-loop level of the parallel programming. This report shows the step-by-step parallel programming method and the parallel performance of the molecular dynamics code on VPP500 and Paragon.

Keywords: Numerical Simulation, Molecular Dynamics, Parallel Programming, Vector Programming, Parallelization, Vectorization, VPP500, Paragon, Two Phase Method

---

* Cooperative Staff, Fujitsu Ltd.
** Cooperative Staff, CRC Research Institute, Inc.

目　　次

Contents

1. 初めに

分散メモリ型並列計算機におけるプログラミングでは，並列化率と並列オーバーヘッドが問題となる．並列化率とは，単一のプロセッサで実行した実行時間の何割の部分を複数のプロセッサに分担できるかと言うことである．この値が並列性能を決める．例えば，並列化率 90%ではプロセッサ数を無限にしても最大並列性能は 10 倍である．並列計算機で 100 倍の性能を得るためには，並列化率は 99%以上である必要がある．並列オーバーヘッドは，プロセッサ間の通信とロードバランスの乱れにより生ずる．分散メモリ型並列計算機では，各プロセッサで計算した計算結果を異なるプロセッサで使用するため，プロセッサ間で計算結果を授受する通信が発生し，それは通常メモリアクセス速度より一桁以上遅い．またプロセッサ間で計算の分担が不均等になると，待ち状態のプロセッサができ，ロードバランスが乱れる．

並列プログラミング（以後，「並列化」と呼ぶ）を分散メモリ型並列計算機で行うためには，各プロセッサが長い時間独立に計算する部分（大粒度）をロードバランスを考慮して作り，通信時間を減らすことが必要である．このことは，複数のループ，ルーチン間に跨り，通信をしない独立した計算をするプログラムを要求する．このような考慮をしていない単一プロセッサの計算機のために書かれたプログラムは，並列化により，プログラムの至る所で通信が生じる可能性を持つ．このことは簡単な例から直ちにわかる．Fig.1.1 は，do ループの計算に a(i+1)のデータが使用されている．a は 100 個の配列であり，2 個のプロセッサに分割して置く（データ分割と呼ぶ）．ここで do 1,99 を 2 個のプロセッサに分担（手続き分割と呼ぶ）した場合，i=50 の計算でプロセッサ 1 はプロセッサ 2 の値を必要とし，通信が発生する．このような計算はプログラムの至る所にあり，通信はプログラム中の各所で発生する．そして，それら通信の全体の処理時間に占める割合は無視できない．

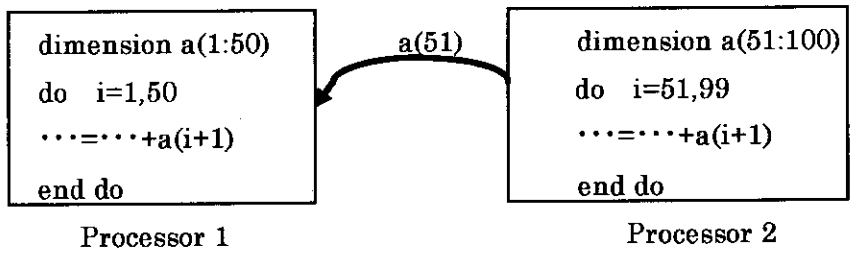

Fig. 1.1 Typical communication pattern of parallelized program

通信を減らして大粒度を得るためには，計算する問題の持つ並列性を積極的に利用する．さらに，問題をコード化したプログラムに直した時，この並列性を使うためには問題のモデル化方法，数値計算スキーム，アルゴリズム，コーデングに並列性を持たせる必要があ

る．また分散メモリ型並列計算機では，通信時間に比べて十分に大きい粒度をつくることが重要である．粒度の大きさはメッシュサイズのような問題の規模に依存するため，通信を減らす時は問題を規定する計算パラメータを考慮する必要がある．

このような状況下で容易に並列化を行うためには，粒度と通信時間の関係を見極めながら，徐々に粒度を大きくし，通信時間を減らしていきたい．通常の並列化は，このような段階的な手続きを経ず，一度にプログラム全体の手続き分割とデータ分割を行う．理由は，逐次計算部分を残しておくと，データ分割したデータを使う場合，通信が発生するためである．この問題は，「２フェーズ法」と呼ばれる並列化方法により回避できる［1］．２フェーズ法は，データ分割をしない所から出発し，手続き分割と計算結果の unify 転送をセットとする並列化方法で，プログラムの変更箇所を局所化し，並列化過程において段階的に並列化率を向上し，通信時間を減らしていくプログラミングを可能にする．またこの方法により，数値シミュレーションに存在するコードレベルの並列性を系統的に利用することが可能となり，既存のプログラムの並列化が容易となる．

そこで本報告書では，分子動力学 MD(Molecular Dynamics)法を用いた既存のシミュレーションコード（以後，MD コードと呼ぶ）を２フェーズ法を用いて並列化し，並列化過程において得られる性能と並列化内容の関係を議論する．並列計算機は，並列性能が粒度と通信時間の相対性能で決まるということから，ベクトル並列計算機の VPP500，スカラ並列計算機の Paragon を使用した．ベクトル計算機は単一プロセッサの処理性能が高いため，粒度が小さくなる傾向がある．スカラ計算機はそれより一桁大きい粒度である．通信性能は，ネットワークのトポロジと，バンド巾により決まるが，VPP500 はネットワークの結合が密であるクロスバネットワークを，Paragon はネットワークの結合が比較的疎なメッシュネットワークを用いている．密にネットワークを結合するトポロジほど，プロセッサ間のデータ転送時間が一律となる．一方，疎にネットワークを結合するトポロジは，目的のプロセッサにデータを送るまでトポロジに従ってデータを経由する必要が生じ，データ転送時間がばらつく．また，複数のプロセッサを経由することは，プロセッサ数の増加と共に通信時間が増す．これらの特徴が並列化した MD コードの性能に現れることが予想される．

本報告書は，２章で並列化に用いた MD コードとその並列性を述べる．３章で２フェーズ法の適用方法を述べる．４章で並列化手順を述べる．５章で VPP500 における並列化について述べる．６章で Paragon における並列化について述べる．７章では並列化に対する２つの計算機の性能を示し，並列性能が得られる計算パラメータの範囲について言及する．８章で議論とまとめを行う．

2．MD コードと並列性

MD コードは，モデル粒子の運動をニュートンの運動方程式を解くことによって記述する．粒子間の力はモデルポテンシャルによって近似され，各々の粒子は自分以外の全ての粒子からの力の影響を受けるため，力の計算回数は粒子数の自乗に比例する．その結果，全計算時間における力の計算の占める割合いが大きくなり，この部分を高速計算する必要が生ずる［2］．一方，実際の MD コードでは力の及ぶ範囲を限定する遮蔽距離を導入して計算量の軽減を図るため，短距離力の計算では力の計算回数は粒子数に比例するようになる［3］．その結果，力の計算以外の部分の計算時間が相対的に増え，この部分の高速計算がさらに必要となる．このことは，力の計算以外の部分の並列化が必要になることを意味する．本章では並列化対象 MD コードの構造を示し，その並列性を調べ，並列化の可能性について述べる．

2.1 計算内容

ここで取り扱う MD コードは、２次元の矩形領域内でのアルゴン原子の熱伝導状態，対流渦の巨視的挙動を解析する［4］．mをアルゴン原子の質量，$\mathbf{v}_i$を i番目のアルゴン原子の速度，$\mathbf{r}_i$を位置座標，$\mathbf{F}_i$を i番目の原子に働く力であるとすると，時刻 t における運動方程式は次のように書くことができる．

$$\frac{d\mathbf{v}_i(t)}{dt} = \mathbf{F}_i(t)/m + \mathbf{g},\ \mathbf{v}_i(t) = \frac{d\mathbf{r}_i(t)}{dt} \tag{2.1}$$

ここに，$\mathbf{g}$は重力加速度である．力$\mathbf{F}_i$は２体間ポテンシャル$\phi(\mathrm{r})$より次のように書ける．

$$\mathbf{F}_i = -m\sum_{j\neq i}\frac{\partial\phi(r_{ij})}{\partial\mathbf{r}_i} \tag{2.2}$$

ここに，r_{ij}はi番目の粒子とj番目の粒子との距離$|\mathbf{r}_i - \mathbf{r}_j|$である．
モデルポテンシャル$\phi(\mathrm{r})$は Lennard-Jones ポテンシャルである．式(2.3)にこれを示す．
ここに，$(\mathrm{n})_{\mathrm{LJ}}$=12，$(\mathrm{m})_{\mathrm{LJ}}$=6，$\varepsilon$ と σ はそれぞれエネルギー及び長さの次元を持つ量である．

$$\phi(r) = 4\varepsilon\left\{\left(\frac{\sigma}{r}\right)^{(n)_{\mathrm{LJ}}} - \left(\frac{\sigma}{r}\right)^{(m)_{\mathrm{LJ}}}\right\} \tag{2.3}$$

取り扱う系は，Fig.2.1 に示す２次元の矩形領域で，温度差がある拡散反射面で底面と上面を，鏡面反射面で左右をかこまれでいる．

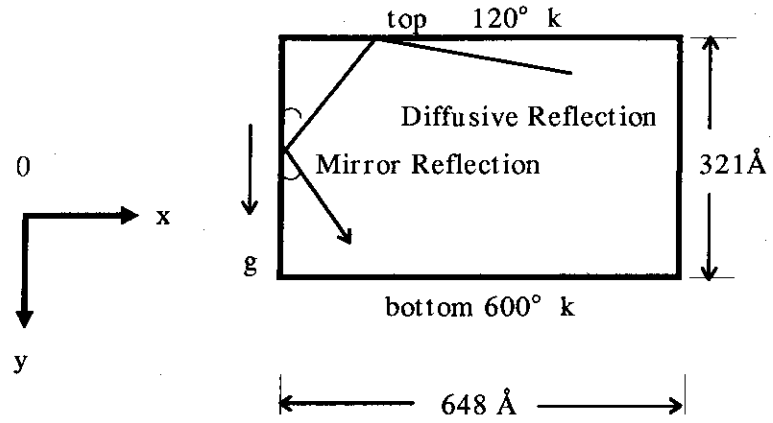

Fig.2.1 Computing conditions

系を現すマクロな物理量は，矩形領域をサンプリングセルに分割し，セル内の平均物理量として定義する．セルの番号をkとすると，各セルの物理量の総和 A_k の平均物理量は式(2.4)で現すことができる．

$$\langle a_k \rangle_t \equiv \frac{1}{t - t_0} \int_{t_0}^{t} \frac{A_k(s)}{N_k(s)} ds \tag{2.4}$$

ここに，N_k は時刻 s におけるセル k 内の粒子数である．物理量の総和 A_k は，時刻 s におけるセル k 内の運動量の総和又はエネルギの総和である．$\langle \cdots \rangle_t$ は，t_0 から t 間の時間平均である．

2.2 離散化

運動方程式(2.1)の離散化は，改良型 Verlet の方法[3]を用いて，次のように表わす．

$$x_i(t+\Delta t) = x_i(t) + v_{xi}(t)\Delta t + 0.5 a_{xi}(t)\Delta t^2 \tag{2.5}$$

$$y_i(t+\Delta t) = y_i(t) + v_{yi}(t)\Delta t + 0.5(a_{yi}(t) + g_y)\Delta t^2 \tag{2.6}$$

$$v_{xi}(t+\Delta t) = v_{xi}(t) + 0.5(a_{xi}(t+\Delta t) + a_{xi}(t))\Delta t \tag{2.7}$$

$$v_{yi}(t+\Delta t) = v_{yi}(t) + \{0.5(a_{yi}(t+\Delta t) + a_{yi}(t)) + g_y\}\Delta t \tag{2.8}$$

ここに a_{xi}，a_{yi} は式(2.2)を m で除したもの，g_y は重力加速度である．

力の計算には，ブックキーピィング法[3]を用いる．この方法は，粒子 i に対し α r_{cut} 内の粒子番号を記述した表を作成し，時間が経過するに従って粒子の位置の変化がある範囲を越えると再作成する(Fig.2.2 参照)．力の計算では，表に記載された α r_{cut} 内の粒子の距離を計算し，r_{cut} 内の粒子か否かを判定し，r_{cut} 内の粒子のみ式(2.3)を一階微分した式より力の計算を行い，計算量を軽減する．通常，$(n)_{LJ}=12$，$(m)_{LJ}=6$ に対し，r_{cut} は 3 σ を用いる．またこの MD コードでは，α は経験的に 4 を用いている．

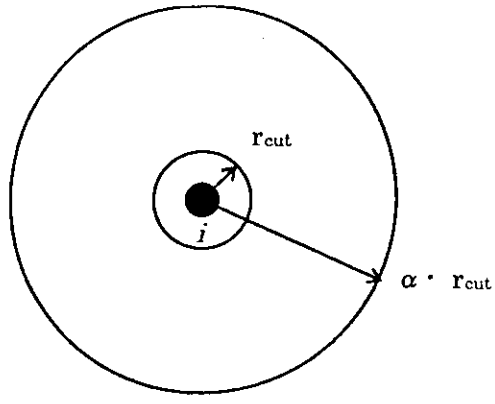

Fig.2.2 The bookkeeping method uses radius α r_{cut} to book particles. Force to the particle i is computed within cut-off radius r_{cut}.

物理量の計算，式(2.4)は，時間ステップ Δt，時間積分回数 I_s を用いて，次のように書ける．

$$\langle a_k \rangle_t = \frac{1}{\Delta t \cdot I_s} \sum_{s=1}^{Is} \left(\frac{A_k}{N_k} \right)_s \tag{2.9}$$

2.3 計算パラメータ

矩形領域は幅 648 Å、高さ 321 Å である。底面は 600° K、上面は 120° K であり，温度勾配は線形である (Fig.2.1)．矩形領域のサンプリングセル数は 40×20 である．粒子の初期条件は，空間的には等間隔配置で 9 粒子/セル，速度は Maxwell 分布である．粒子即ちアルゴン原子の数は 7200 個である．遮蔽距離 r_{cut} は 3 σ．α は 4 である．また，タイムステップは 2000 回／サンプリング数．サンプリング数は 6 回である．

2.4 プログラム構造

プログラムのフローチャートを Fig.2.3 に示す．サブルーチン init では，粒子の初期位置を計算する．table では、ブックキーピィング法で使う表を作成する． maxwell では，

初期値として乱数から各粒子の速度を計算する．force で table で作成した表を利用して力を計算する．pcross と chkbnd で境界面を判定し，pmoves で粒子の移動と反射を行う．更に，拡散反射の場合一様乱数が用いられる．物理量計算は propcel で行う．
main プログラムでは，時間積分のくり返しの制御，サンプリングの制御を行う．また，離散化された運動方程式，式(2.5)～(2.8)の計算を main で行う．

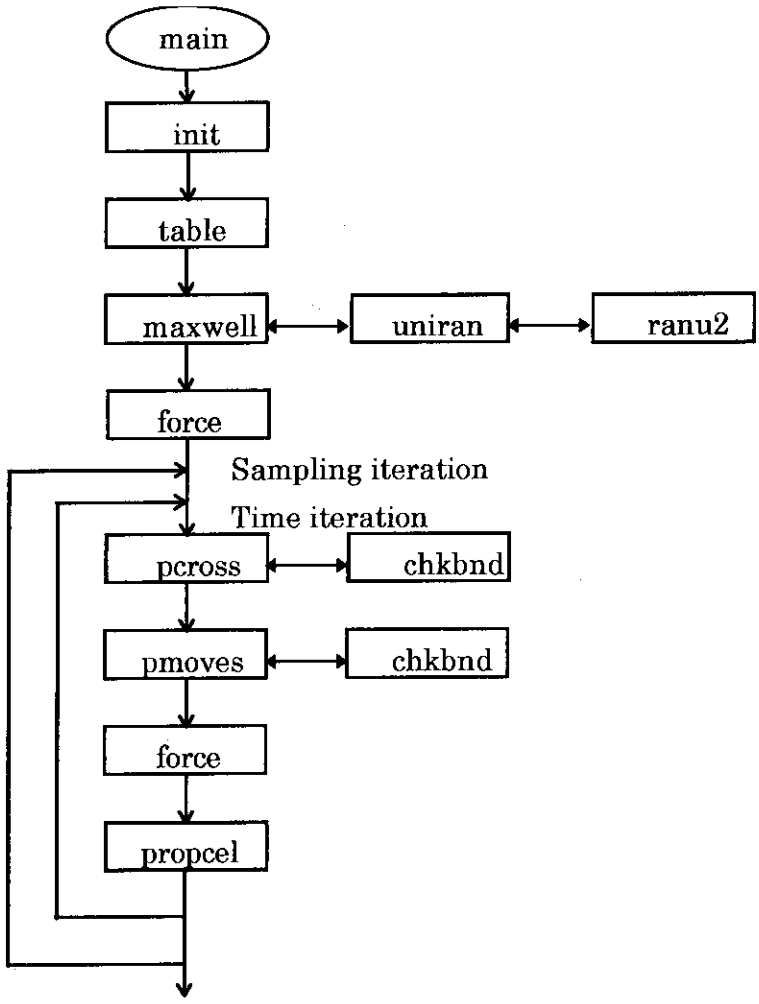

Fig.2.3 Flowchart of MD code

2.5 並列性

離散化された運動方程式(2.5)～(2.8)は陽解法である．従って，これらの式は粒子毎に並列性を持つ．これらの計算は，main プログラムの一部とした記述されている．

式(2.2)の力の計算は，総和計算を除き粒子毎に並列性を持つ．これらの計算はサブルー

チン force で計算される．

粒子の位置の判定と，粒子の移動，境界上の反射の計算は粒子毎に並列性を持つ．これらの計算は，サブルーチン pcross, chkbnd, pmoves で行われる．

ブックキーピィング法の表の計算は，粒子間の距離の計算に基づく表の作成であり，粒子毎に並列性を持つ．この計算は，サブルーチン table で行われる．

式(2.9)の物理量の計算は，総和計算を除き，粒子毎に並列性を持つ．この計算はサブルーチン propcel で行われる．

これらのことから，ここで取り上げた MD コードのモデルと数値計算スキームは，力と物理量の総和計算を除いて並列性を持つ．並列計算機上でこれらの並列性を利用するためには，この計算スキームの並列性を維持した状態でプログラミングを行うことが必要である．その場合，プログラムに使用するアルゴリズムやコーディング方法がこの計算スキームの並列性を保つものに限定される． モデルと数値計算スキームの並列性と，計算アルゴリズムや，コーディングとの並列性が一致しないプログラムは，大幅な書き換えが必要になる．これについては，7.3 節で述べる．

3. 段階的並列化方法：2フェーズ法

分散メモリ型並列計算機では，3 つのプログラミング機能を用い並列化を行う．「手続き分割」は，計算を各プロセッサに割り当ててターンアラウンド時間の短縮を実現する．「データ分割」は，データを各プロセッサに分配することで大規模な問題を扱えるようにする．「データ転送」は，計算に必要なデータをプロセッサ間で授受する．Fig.3.1 にこれらの機能を示す．

2 フェーズ法[1]では，並列化を 2 つのフェーズ「手続き分割とデータ転送」と「データ分割」に分ける．またデータ転送として，「unify 転送」と呼ぶ計算結果を共有するデータ転送を導入する．これによりデータ分割を後回しにすることができ，並列化する場所以外でのデータ転送の考慮をしなくてすむ．それは並列化対象以外の部分でプログラムの変更をしなくてすむことを意味し，データ分割を行った場合と比較すると，並列化作業が容易となる． 2フェーズ法は，プログラムの変更範囲をローカライズし，要求される性能に従って並列化範囲を順次広げることができる，段階的並列化方法である．

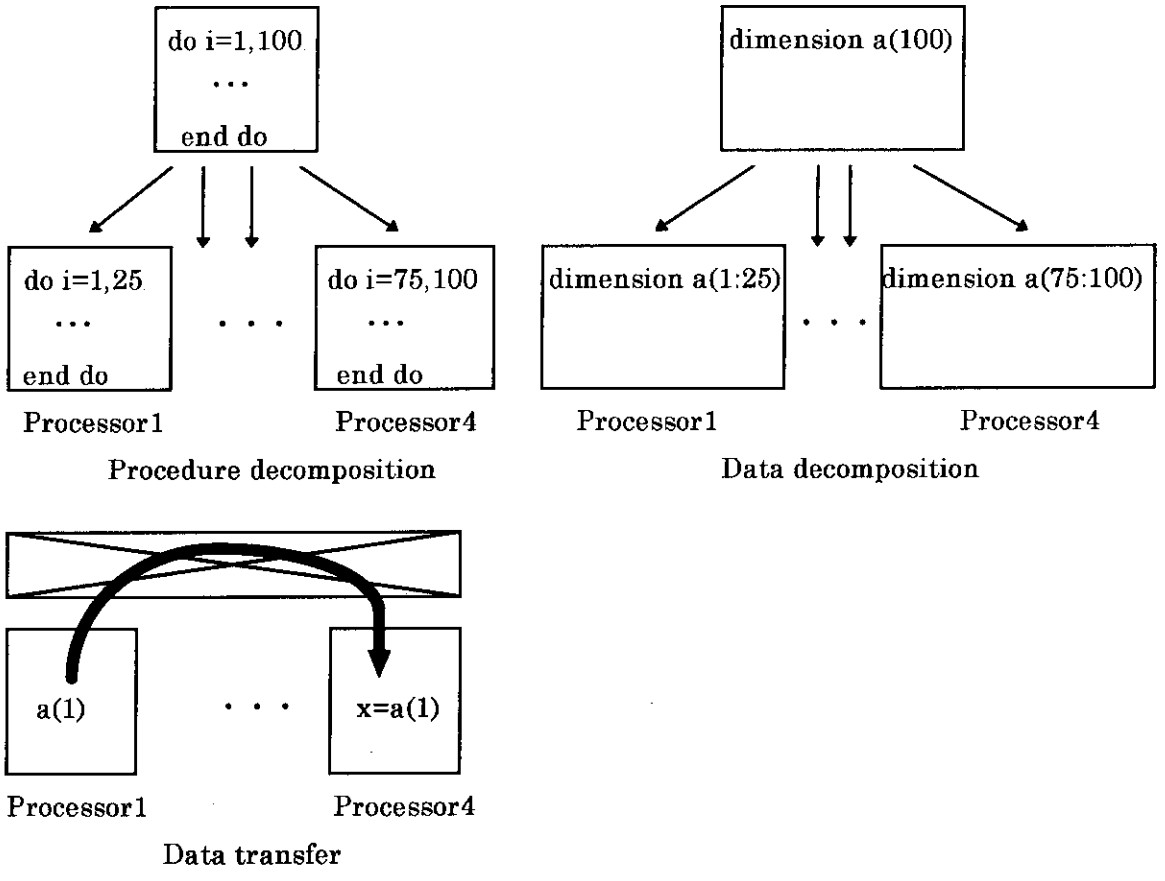

Fig. 3.1 Three programming functions for parallel programming.

3.1 フェーズ１（手続き分割，unify 転送）

フェーズ１では並列化対象部分の手続き分割と，計算結果を共有する unify 転送を行う．unify 転送とは，Fig.3.2 のように自プロセッサの計算結果を他の全てのプロセッサに転送することである．unify 転送の後，全てのプロセッサは同じ計算結果を持ち，全てのプロセッサが同じ計算をするため，プロセッサ間での通信が不要となる．

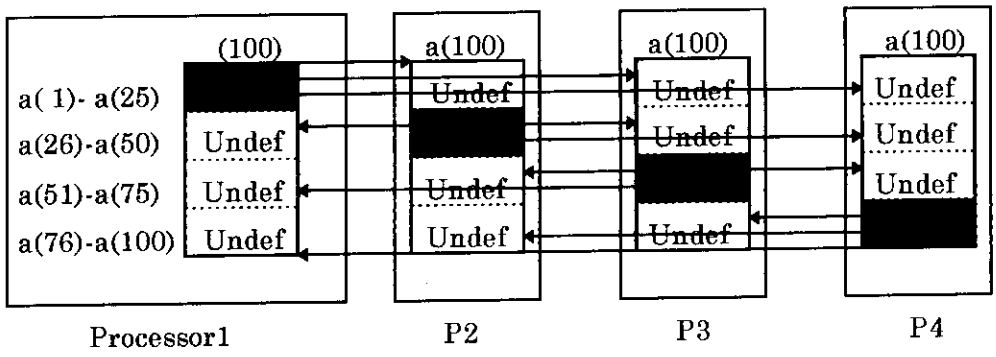

Fig. 3.2 Schematic view of unifying data between processors.
The shadowed part of each processor is a result of procedure decomposition.

フェーズ１のプログラミング例を Fig.3.3 に示す．Fig.3.3b は VPP500 の Fortran 言語拡張 VPP FORTRAN77 EX/VPP[5] によるプログラミング例を，Fig.3.3c は Paragon の Fortran とメッセージパッシングライブラリ[6]による例である．これらの例では unify 転

```
do j=1,jmax
  do i=1,imax
    ・・・
    b(i,j)=・・・
  end do
end do
```

Fig. 3.3a Original program running on uni-propcessor.

```
      dimension wb(*,*)
        [wb←0]
!xocl spread do
      do j=1,jmax
        do i=1,imax
          ・・・
          wb(i,j)=・・・
        end do
      end do
!xocl end spread sum(wb)
        [b←wb]
```

Fig. 3.3b Unifying data with VPP FORTRAN77 EX/VPP.

```
[Np:Number of processors]
[iam: Processor ID]
      dimension wb(*,*)
        [wb←0]

      do j=iam,jmax,Np
        do i=1,imax
          ・・・
          wb(i,j)=・・・
        end do
      end do
      call gdsum(wb,wk)
        [b←wb]
```

Fig. 3.3c Unifying data with Paragon library.

送の機能を、 VPP500は，!xocl end spread の sum(bw)の部分のプロセッサ間の総和計算，Paragonはメッセージパッシングライブラリの gdsum で実現する．総和を取るため新たにワーク配列 wb を設け，総和計算後オリジナル配列 b に wb を代入する．

3.2 フェーズ2（データ分割）

フェーズ2は，分割が簡単な大きな配列を選んでデータ分割し，計算に必要なメモリを確保する．Fig.3.4 に VPP FORTRAN77 EX/VPP でデータ分割した例を示す．!xocl processor p(4)よりプロセッサ数を4と宣言する．!xocl local c(:,/(p))より配列 c を2次元目で4つに分割する．例では，このデータ分割により各プロセッサは 1000×7500 要素の領域を削減できる．配列 b が整合配列の場合，これを common 配列に変更し，コンパイル時にその大きさを決定して，データ分割する．

MD プログラムに対するフェーズ2の適用方法は，5.4節，6.4節において述べる．

```
dimension a(10000000)
   m=1000
   n=10000
   call int(a(1),a(m+1),m,n)
   ・・・
end
subroutine int(b,c,m,n)
dimension b(m),c(m,n)

   ・・・
end
```

Fig. 3.4a Original program running on uni-propcessor.

```
!xocl processor p(4)
      parameter(m=1000,n=10000)
      dimension a(m)
      common/D/c(m,n)
!xocl local c(:,/(p))
      call int(a)
      ・・・
      end
      subroutine int(b)
      parameter(m=1000,n=10000)
      dimension b(m)
      common/D/c(m,n)
!xocl local c(:,/(p))
      ・・・
      end
```

Fig. 3.4b Data decomposition with VPP FORTRAN77 EX/VPP.

3.3 unify 転送の最適化

unify 転送を多用すると，全体の計算時間に対し転送時間が無視できなくなる．この問題

は，次に述べる２つの方法によって解決できる場合が多い．

(1)不要な unify 転送の削除

プログラムの構造によっては， unify 転送が不要な場合がある．それは，並列化した do ループの次の do ループの中で他のプロセッサが計算した値を使用しない場合である．この unify 転送不要の例を Fig.3.5 に示す．b の unify 転送は，次の並列化 do ループが同一プロセッサ内のデータのみを参照するため不要であり，削除することができる．

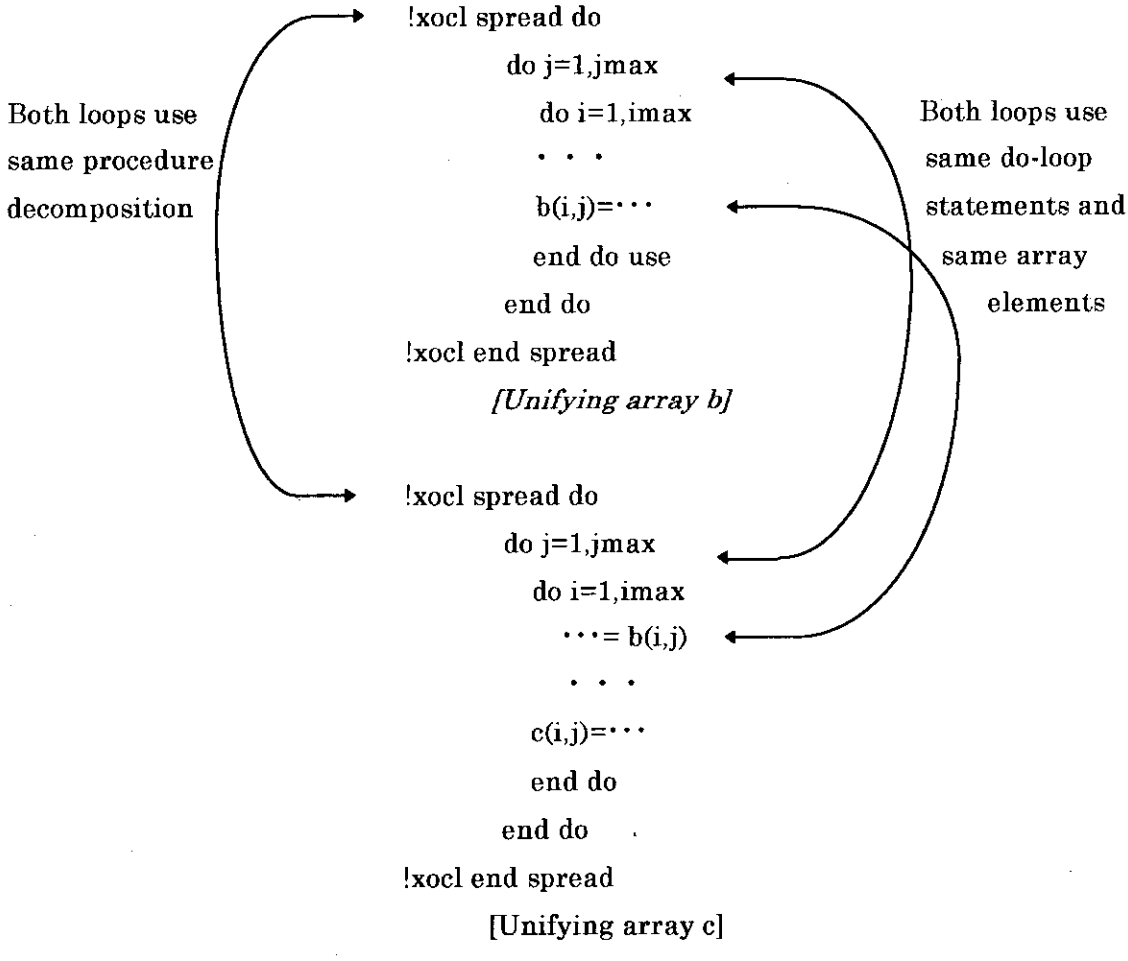

Fig. 3.5 Redundant procedure:*[Unifying array b]*

(2)部分 unify 転送

総和計算を利用した unify 転送は，定義した配列の要素全てを転送する．従って，配列の要素数が多くなった場合，転送量が増え，転送時間も増える．そこで次の計算に必要な部分だけを転送することにより転送の軽減を図る．例えば VPP FORTRAN77 EX/VPP で

は，袖と呼ばれる部分のみの転送ができる overlapfix 文が有り，部分 unify 転送に利用することができる．

3.4 並列デバッグと２フェーズ法

並列デバッグにおいては，バグが並列化したため発生したか否かを知ることが重要である．その場合にも２フェーズ法は有効である．理由は，フェーズ１で並列化した箇所は，逐次実行にもどし両者の計算結果を比べること[7]が容易にできるためである．また，不要な unify 転送を削除した際生じる，必要な転送を忘れたことによるバグは，通常の unify 転送から徐々に範囲を広げることによって，発生箇所を特定できる．これは，一度に並列化を行う従来の並列化方法ではできないことである．これにより並列化作業におけるデバッギィングの占める割合を大幅に削減できる．プログラミング上のポイントは，逐次プログラムの部分，消去した unify 転送部分をコメントにして残しておくことである．

4. コードの並列化手順

並列化性能を決める要因は，単一プロセッサ性能，並列化率，並列オーバーヘッドの 3 つに大別できる[8]．並列化はこれら 3 つの要因を考慮しながら行う．その手順は，プログラム分析，単一プロセッサ性能向上，並列化率向上，並列オーバーヘッド削減の順で行い，必要ならこれを繰り返す．

4.1 プログラム分析

性能向上を達成するためには，まず時間を費やす計算箇所を探し出す必要がある．そのためには，プログラムを実行して各ルーチンが占める計算時間の割合（以後，時間コスト分布と呼ぶ）の計測を行う．計測のツールとしては，各システムが用意した経過時間を示す時計，サブルーチン等の実行頻度を調べるサンプラ[9]等を利用する．計測結果を分析して，時間を費やす計算箇所の単一プロセッサ性能向上と並列化を行う．

4.2 単一プロセッサ性能向上

各々のプロセッサが持つ，複数の演算器の使用効率向上を行う。ベクトル計算機の場合，ベクトル化率の向上，ベクトル長の向上，メモリアクセス効率向上，do ループ内の演算密度の向上を行う[10]．スカラ計算機の場合，do ループ内の演算密度の向上，キャッシュの有効利用を行う．

4.3 並列化率向上

並列化率 Rp を Fig.4.1 のように単一プロセッサの実行時間における，並列計算できる部分の比と定義し，単一プロセッサの性能を基にしたプロセッサ数 Np の時の性能の比を P とする時，最大並列性能比 P_m を式(4.1)から予測することができる．

$$P_m = \frac{1}{1 - Rp + Rp / Np} \tag{4.1}$$

ここに，Np はプロセッサ数である．

また使用したプロセッサ数に対する並列効率 Ep は，式(4.2)のように定義される．

$$Ep = \frac{P}{Np} \tag{4.2}$$

ここに性能は，Flops 値又は計算時間の逆数である．今 $P = P_m$ とすると，並列化率に依存した最大並列性能比，最大並列効率を算出することができる．例えば，Rp=0.9，Np=10 で並列計算をする場合，P_m＝5.3 倍で，Ep ＝53%である．そのままの並列化率で Np=100 とすると，P_m＝9.2 倍で，Ep ＝9.2%である．これは，投入したプロセッサ数に見合った並列化率が必要であることを示す．従って，予めプログラム中で並列化できる部分を調べて並列化率を予測し，必要な並列性能比とそれを達成するプロセッサ数を検討する必要が

ある．このために，$P = P_m$ として式(4.2)に式(4.1)を代入し，Np を求める式に直すと便利である．これを式(4.3)に示す．

$$Np = \frac{1/Ep - Rp}{1 - Rp} \tag{4.3}$$

尚，並列化率は単一プロセッサの性能によって変化する．単一プロセッサにおける性能向上の結果，Fig.4.1 の$(Tt)_1$が減り，それ以上に$(tp)_1$が減ればその部分の全体に占める割合が減り，並列化率は下がることに注意する必要がある．

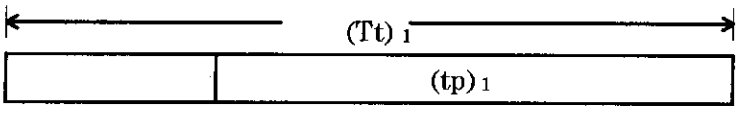

Fig. 4.1　Definition of parallelization ratio : Rp　$(tp)_1 / (Tt)_1$, where $(Tt)_1$ is whole uni-propcessing time of a code, and $(tp)_1$ is time of parallelizable part of the code.

4.4 並列オーバーヘッド削減

並列オーバーヘッドには大きく分けて 2 つのものがある．一つはプロセッサ間の通信に関するもの．もう一つはロードバランスである．

通信に関するものは，大別すると次の処理に分けられる．

・データ転送

・同期

・タスク生成消滅

データ転送時間の削減方法は，第一に高速通信ライブラリを使うこと．第二に，不要な通信の削除である．

同期は，計算を正しく進めるために必要な所のみで用いる．特にソフトウェア同期は，実行時間が長いため，注意が必要である．

タスク生成消滅時間は，データ領域を含んだプログラムを各プロセッサに送るため，時間がかかる．VPP500 のように生成消滅がプログラムで書ける（!xocl parallel region, !xocl end parallel）場合は，なるべく 1 回で済むようにする．

次にロードバランスが損なわれる主な原因を示す．これらは，個別の対応が必要である．

・I/O がある場合

・異なった種類の計算で並列化を行う場合

・do ループの回転数がプロセッサ数で割り切れない場合

・do ループの中に if がある場合

・多重ループで内側ループの回転数が外側ループのインデックスに依存して変化する場合

多重ループで内側ループの回転数が外側ループのインデックスに依存して変化する場合は，手続き分割にcyclic分割を用い，ロードバランスを確保する．5.3節，6.3節でこれを示す．

5. VPP500における並列化

VPP500は，1.6Gflopsのピーク性能と256MBのメモリを持つPE(Processing Element)を，送受信各々400MB/secのクロスバネットワークで接続した分散メモリ型並列計算機である．並列化には，データを複数のプロセッサに分散配置しこれらを並列に処理する「データパラレル」タイプの言語VPP FORTRAN77 EX/VPPコンパイラ[5]を用いる．コンパイラディレクティブ!xoclで始まるこの言語は，通常のFortranコンパイルではコメント文となる．このため，逐次計算機とVPP500両方で実行可能なプログラムを書くことができる．VPP500では，ベクトル性能と並列性能を十分に引き出すため，十分なベクトル長と大粒度になるようにプログラミングする．そこで，ネストした多重doループでは異なったdoループにベクトル化と並列化を割り当てる．

5.1 プログラム分析

プログラム分析では，単一のプロセッサでプログラムを実行し，時間コスト分布を調べる．Fig.5.1はVPP500のツール「サンプラ」で調査したコスト分布である．サンプラは特定の時間間隔で実行中のルーチン名を調べ，その回数を積算して時間コスト分布を表現する．

```
Status                          : Serial
Number of Processors            : 1

Type                            : cpu
Interval (msec)                 : 20

Synthesis Information
      Count|   Percent|      VL|   Name
      83212|      77.3|      72|   force_
      11263|      10.5|     963|   table_
       6897|       6.4|       -|   chkbnd_
       2916|       2.7|       -|   pmoves_
       1684|       1.6|    1424|   propcel_
       1533|       1.4|       -|   pcross_
        166|       0.2|    1986|   MAIN__
          7|       0.0|       -|   ranu2_

     107678|          |     114|   TOTAL
```

Fig.5.1 Time cost distribution of original MD code

図は 20msec の時間間隔でルーチン名をサンプリングし，測定した時間コスト分布である．図中， Name はルーチン名， Count は積算回数，全体の積算回数を 100%とした時の各ルーチンのサンプリング回数をパーセントで表わしたものが Percent に示される．さらに，各ルーチンの平均ベクトル長(一回のベクトル計算で扱う要素数．do ループの回転数とは異なる．) が VL に示される．Fig.5.1 は，このプログラムの主要時間コストルーチンが force, propcel, table, pcross, pmoves, chkbnd であることを示す．

5.2 単一プロセッサにおける最適化

VPP500 はベクトル並列計算機であるため，1 プロセッサでの最適化はベクトル化が中心となる．最適化の後，ベクトル化のコンパイルとしては frtpx -Ad -Ne,50 -Ei -L/opt/px/lib -lls2vp *.f を，ベクトル並列化のコンパイルとしては frtpx -Wx -Ad -Ne,50 -Ei -L/opt/px/lib -lls2vp *.f を用いる． -L/opt/px/lib -lls2vp は科学用サブルーチンライブラリ SSL II [11] の一様乱数ライブラリ ranu2 を使用するためのオプションである．-Wx は並列モードで実行するためのオプションである．

(a)サブルーチン force のベクトル化

Fig.5.2a はオリジナルプログラムのコンパイルリストである．ベクトル化の情報はこのリストの左側に示されている． do270 はこのサブルーチンの計算時間の殆ど全てを占める．その内側のループの do271 の左のベクトル化表示記号 m は，このループ中の演算がベクトルとスカラーの演算混合になっていることを表す．この原因は，コンパイルリストの文末にある vectorization message によってわかるように，配列 fx と fy が isn 番号 26 と 29 において回帰参照を行う可能性があるためである．回帰計算の可能性とは，do271 のループの始めにおいて j が配列 itab を参照して決定され，これを用いて fx と fy が計算されている一方、isn 番号 26 と 29 で i を用いて計算を行うため，do271 が一回実行される間に i と j が同じ値になる可能性のことである．尚，do 文の左のベクトル化表示記号が v の場合，そのループは完全にベクトル化されている．また s の場合，その do ループはスカラ計算で実行される．

```
isn
00000001                subroutine force(rcut2)
00000002                include './inc2'
00000003          c
00000004        v         do   225   i=1,n
00000005        v           fx(i) = 0.0
00000006        v           fy(i) = 0.0
00000007        v     225   continue
00000008          c
00000009        s         do   270   i=1,ipmax
00000010        s           if(icol(i).ne.0) then
```

```
00000011        s               xi = x(i)
00000012        s               yi = y(i)
00000013        m           do  271  k=1,icol(i)
00000014        v               j    = itab(i,k)
00000015        v               xx = xi - x(j)
00000016        v               yy = yi - y(j)
00000017        v               rd   = xx * xx + yy * yy
00000018            c
00000019        v               if ( rd   .gt. rcut2 ) goto 271
00000020            c
00000021        v               rdr    = 1./rd
00000022        v               rd3    = rdr**3
00000023        v               rd4    = rdr**4
00000024        v               rd     = (rd3-0.5)*rd4
00000025        v               fxx     = xx       * rd
00000026        s               fx(i)   = fx(i) + fxx
00000027        s               fx(j)   = fx(j) - fxx
00000028        v               fyy     = yy       * rd
00000029        s               fy(i) = fy(i) + fyy
00000030        s               fy(j) = fy(j) - fyy
00000031            c
00000032        v       271 continue
00000033        s               endif
00000034        s       270 continue
                                . . .
fortran77 ex/vp     vectorization messages: program name(force)
jpc2217i-i isn:00000026                 Array fx cannot be vectorized because recursive
reference takes place.
jpc2207i-i isn:00000026 - 00000027  Array fx cannot be vectorized because variable j in
subscript expression is defined in this DO loop.
jpc2217i-i isn:00000029                 Array fy cannot be vectorized because recursive
reference takes place.
jpc2207i-i isn:00000029 - 00000030  Array fy cannot be vectorized because variable j in
subscript   expression is defined in this DO loop.
```

Fig.5.2a　VPP500 compile list of original force

一方，この計算が作用反作用の計算のため，回帰計算とはならないことはわかっているので，回帰計算で無いことをコンパイラに示すことによってベクトル化することができる．Fig.5.2b の*vocl loop,novrec はこのための指示行である．また，isn 番号 35 と 40 の計算は別の変数を用いて記述し計算結果を do271 終了直後に代入することにより，独立した計算であることを明示する．これら 2 つの修正により do271 は完全ベクトル化できる．Fig.5.2b の do271 の左のベクトル化表示記号 v が，完全ベクトル化になったことを示す．

さらにこのベクトル化では itab のインデックスを逆転させた．これはベクトル計算において連続アクセスを行い，メモリアクセスの高速化を図るためである． isn 番号 21 と 22 にこれを示す．

```
isn
00000001                subroutine force(rcut2)
00000002                include './inc2'
00000003          ci
00000004                include './inc3'
00000005          c
00000006        v       do  225  i=1,n
00000007        v         fx(i) = 0.0
00000008        v         fy(i) = 0.0
00000009        v   225  continue
00000010          c
00000011        s       do  270  i=1,ipmax
00000012          cd     if(icol(i).ne.0) then
00000013          ci
00000014        v         xi = x(i)
00000015        v         yi = y(i)
00000016        v         fxi$=0.d0
00000017        v         fyi$=0.d0
00000018          ci
00000019          *vocl loop,novrec
00000020        v       do  271  k=1,icol(i)
00000021          cd     j   = itab(i,k)
00000022        v         j   = itab(k,i)
00000023        v         xx = xi - x(j)
00000024        v         yy = yi - y(j)
00000025          c
00000026        v         rd  = xx * xx + yy * yy
00000027          c
00000028        v         if ( rd  .gt. rcut2 ) goto 271
00000029          c
00000030        v         rdr   = 1./rd
00000031        v         rd3   = rdr**3
00000032        v         rd4   = rdr**4
00000033        v         rd    = (rd3-0.5)*rd4
00000034        v         fxx    = xx      * rd
00000035          cd      fx(i)   = fx(i) + fxx
00000036          ci
00000037        v         fxi$   = fxi$ + fxx
00000038        v         fx(j)   = fx(j) - fxx
00000039        v         fyy    = yy      * rd
00000040          cd      fy(i) = fy(i) + fyy
00000041          ci
00000042        v         fyi$   = fyi$ + fyy
00000043        v         fy(j) = fy(j) - fyy
00000044          c
00000045        v   271   continue
00000046          ci
00000047        m          fx(i) = fx(i) + fxi$
00000048        s          fy(i) = fy(i) + fyi$
00000049          cd      endif
00000050        v   270   continue
                          . . .
fortran77 ex/vp    vectorization messages: program name(force)
jpc2306i-i isn:00000011              Partial vectorization overhead is too large.
```

Fig. 5.2b VPP500 compile list of vectorized force

(b)サブルーチン table のベクトル化

table の do100 はこのサブルーチンの計算時間のほぼ全てを占める．Fig.5.3a にこれを示す．do100 の左のベクトル表示は m で，force と同様にベクトル計算とスカラ計算の混合となっている．この原因は，if 文中で使用されている icol が回帰参照を行っているとコンパイラが判断したためである．実際に回帰計算は行われていないので，icol をスカラ変数に置き換えて書き直すことにより，ベクトル化が可能になる．これを Fig.5.3b に示す．尚，itab のインデックスは，Fig.5.2b の isn 番号 22 の処理と整合性をとるために入れ替えた．

```
isn
00000001                subroutine table(iclock,rint2)
00000002                include   './inc2'
                         . . .
00000006       s        do 101 i=1,ipmax
00000007          c
00000008       v        xi = x(i)
00000009       v        yi = y(i)
00000010                icol(i) = 0
00000011          c
00000012       m         do 100 j=i+1,n
00000013       v         rx = (x(j)-xi)**2+(y(j)-yi)**2
00000014       v         if(rx.lt.rint2) then
00000015       m            icol(i) = icol(i)+1
00000016       v           itab(i,icol(i)) = j
00000017       v         end if
00000018       v    100  continue
00000019       v        if(imax.lt.icol(i)) imax=icol(i)
00000020       v    101  continue
                          . . .
fortran77 ex/vp    vectorization messages: program name(table)
jpc2306i-i isn:00000006              Partial vectorization overhead is too large.
jpc2217i-i isn:00000015              Array icol cannot be vectorized because recursive
reference takes place.
```

Fig. 5.3a VPP500 compile list of original table

```
00000001                subroutine table(iclock,rint2)
                        . . .
00000007          c
00000008       s        do 101 i=1,ipmax
00000009          c
00000010       v        xi = x(i)
00000011       v        yi = y(i)
00000012          cd     icol(i) = 0
00000013          ci
00000014       v         icol$    = 0
00000015          c
00000016       v        do 100 j=i+1,n
00000017       v         rx = (x(j)-xi)**2+(y(j)-yi)**2
00000018       v         if(rx.lt.rint2) then
00000019          cd        icol(i) = icol(i)+1
00000020          ci
00000021       v            icol$    = icol$   +1
```

```
00000022          cd        itab(i,icol(i)) = j
00000023          ci
00000024      v             itab(icol$,i   ) = j
00000025
00000026          cd        if(inum(i).ge.n) write(6,*) '*i,inum*',i,inum(i)
00000027      v             end if
00000028      v    100      continue
00000029          ci
00000030      v             icol(i)=icol$
00000031      s             if(imax.lt.icol(i)) imax=icol(i)
00000032      v    101      continue
                            . . .
```

fortran77 ex/vp vectorization messages: program name(table)
jpc2306i-i isn:00000008 Partial vectorization overhead is too large.
jpc2310i-i isn:00000016 Since variable icol$ may be used in the inner DO loop, it cannot be vectorized.
jpc2311i-i isn:00000031 - 00000031 Since referencing of variable imax precedes its definition, it cannot be vectorized.

Fig. 5.3b VPP500 compile list of vectorized table

(c)サブルーチン pmoves のベクトル化

このサブルーチンは粒子の位置を系内と境界上で計算するが，計算時間の殆どは系内の計算である．そこで，オリジナルコード Fig.5.4a に対して，Fig.5.4b のように系内と境界上の計算を分割し，系内の計算のみをベクトル化した．

```
isn
00000001                subroutine pmoves(iclock)
00000002                include './inc2'
00000003                do 300 i=1,n
00000004                 men = mencol(i)
00000005                   if( men.eq.0 ) then
00000006                      x(i) = x(i) +(vx(i) + fx(i)*dt2)*dt
00000007                      y(i) = y(i) +(vy(i) +(fy(i) + grav)*dt2)*dt
00000008                   else
00000009                     time =  timcol(i)
00000010                     xt   =  x(i)
00000011                     yt   =  y(i)
                           . . .
00000123                   call chkbnd( men,time,xt,yt,uu,vv, tmrest)
                           . . .
00000132                   vy(i)  = vv
00000133                   endif
00000134             300 continue
                           . . .
```

fortran77 ex/vp vectorization messages: program name(pmoves)
jpc2004i-i isn:00000003 This DO loop is not vectorizable since it contains more than 2 exits.
jpc2101i-i isn:00000003 This DO loop is not vectorizable since it contains an unvectorizable inner loop.

Fig. 5.4a VPP500 compile list of original pmoves

```
isn
00000001                subroutine pmoves(iclock)
                        ・・・
00000004                dimension list$(n)
                        ・・・
00000007                nlist$=0
00000008        v       do 300 i=1,n
00000009        v       men = mencol(i)
00000010        v         if( men.eq.0 ) then
00000011        v             x(i) = x(i) +(vx(i) + fx(i)*dt2)*dt
00000012        v             y(i) = y(i) +(vy(i) +(fy(i) + grav)*dt2)*dt
00000013        v         else
00000014        v             nlist$=nlist$ + 1
00000015        v             list$(nlist$)=i
00000016        v           end if
00000017        v    300  continue
00000018
00000019                    do 9300 ii=1,nlist$
00000020                        i=list$(ii)
00000021                        men   =   mencol(i)
00000022                        time =   timcol(i)
00000023                        xt    =   x(i)
00000024                        yt    =   y(i)
                                ・・・
00000142                        vy(i)   = vv
                                ・・・
00000146            9300 continue
fortran77 ex/vp    vectorization messages: program name(pmoves)
jpc2004i-i isn:00000019                This DO loop is not vectorizable since it contains
more than 2 exits.
jpc2101i-i isn:00000019                This DO loop is not vectorizable since it contains an
unvectorizable inner loop.
```

Fig. 5.4b　VPP500 compile list of vectorized pmoves

(d)サブルーチン pcross のベクトル化

このサブルーチンはサブルーチン chkbnd を呼んでいるため，ベクトル化できない．しかし，コンパイルオプション-Ne,50 -Ei を指定することにより，そのルーチンを呼び場所にインライン展開することによりベクトル化できる．-Ne,50 は同一ファイル内にあるルーチンの内，ライン数 50 行以内のルーチンをインラインの対象にすることを意味する．また，-Ei は呼出し場所に展開したか否かをメッセージで知るためのオプションである．Fig.5.5 にこの様子を示す．call chkbnd の左のベクトル表示が vi になっている．これは、コンパイラによるインライン展開が行われ，結果としてベクトル化したことを示す．do10 左のベクトル化表示記号 v により，このループが完全にベクトル化したことが判る．

```
isn
00000151                subroutine pcross
                         . . .
00000155          c
00000156        v         do   10   i=1,n
00000157        v           uu = vx(i)+fx(i)*dt2
00000158        v           vv = vy(i)+(fy(i)+grav)*dt2
00000159        vi          call chkbnd(men,time,
00000160                  c           x(i),y(i),uu,vv,dt)
00000161        v           mencol(i)=men
00000162        v           timcol(i)=time
00000163
00000164        v      10   continue
00000165
00000166                  return
00000167                  end
```

fortran77 ex/vp　diagnostic messages: program name(pcross)

jwd8101i-i isn:00000159　This subprogram(CHKBND) is integrated.

Fig. 5.5　VPP500 compile list of vectorized pcross

5.3 並列化(フェーズ 1)

単一プロセッサにおける最適化即ちベクトル化により，時間コスト分布が変化する．このためプログラム分析を再度行う．次に 3.1 節で示したように手続き分割を行う．

(a)プログラム分析

Fig.5.6 は単一プロセッサにおける最適化を実施した後の「サンプラ」の時間コスト分布である． 時間コストが上位の順からコードを解析し，2 フェーズ法による並列化の可能性について調べた結果，force, propcel, table は，容易に並列化できることがわかった．この並列化方法については，(b)以下で述べる．

Status : Parallel
Number of Processors : 1

Type : cpu
Interval (msec) : 20

Synthesis Information

Count	Percent	VL	Name
30746	90.5	55	force_
1621	4.8	1871	propcel_
1026	3.0	1490	table_
245	0.7	1934	MAIN__
226	0.7	900	pcross_
93	0.3	1137	pmoves_
5	0.0	1	ranu2_
3	0.0	-	uniran_
1	0.0	-	_start
33966		151	TOTAL

Fig. 5.6 Time cost distribution of vectorized MD code

図は，これら 3 つのサブルーチンを並列化すると並列化率が 98.3%になることを示す．この並列化率でかつ Ep=0.5，0.8，0.9 を式(4.3)に代入し，それを満たすプロセッサ数 Np を求めると，(Ep=0.5,Np=60)，(Ep=0.8,Np=16)，(Ep=0.9,Np=8)となる．現在，日本原子力研究所にある VPP500 のプロセッサ数は 42 台，この内 1 ユーザが常時使えるプロセッサ数

は16台である．従って，これら3つのサブルーチンの並列化で並列効率の値より計算機資源を有効に利用した計算ができるといえる．尚，並列化の際には，式(4.2)が並列オーバーヘッドを考慮していないことに留意する必要がある．

この分析で，プロセッサ数を数十台，数百台に増やす時，Ep は非常に小さくなり，main, pcross, pmoves についても並列化が必要なことがわかる．しかしこれらのルーチンに2フェーズ法を適用すると，計算のオーダと unify 転送のオーダが粒子数 n に比例するため，並列性能が向上しない．これは，計算と転送の最大性能を比較すると直ちに理解できる．この問題を解決するためには，3.3節で述べた不要な unify 転送の削除，部分 unify 転送を行うことが必要となる．2.5節で示した並列性から，これらを行うことが原理的には可能であるが，簡単な机上スタディで，これを行うためには pmoves の拡散反射で使用する乱数について，プロセッサ数を変えても計算結果が一致する乱数発生のアルゴリズムを開発する必要があることがわかる．そしてここから生ずる開発作業は，「簡単に並列化を行う」という本報告書の主旨から外れるため，これらのルーチンの並列化については言及しないことにした．

(b)force の並列化

force は粒子の位置を入力し，力を出力するサブルーチンで，計算は2重ループで行われる．2.5節で示した並列性から，粒子毎の計算は並列に行うことができ，プロセッサに跨る力の総和計算が並列オーバーヘッドとなる．並列化はこの2重ループの外側のループ do270 で行う．

プログラムを Fig.5.7 に示す．手続き分割には cyclic 分割を用いる． cyclic 分割は，図中では!xocl spread do/ip の ip により指示する．これは，ロードバランスを考慮するためである．計算した fx, fy は，!xocl end spread sum(fx), sum(fy)によってプロセッサ間の総和を取りつつ，unify 転送される．このプロセッサ間の総和計算は，ロードモジュール a.out に，次のようなランタイムオプションで総和計算の作業領域の確保を行う．

a.out -Wl,-Pg{num}

ここに，num=要素数×サイズ(Kbyte)である．
尚，このオプションはプロセッサ間の総和計算の性能向上に必須である．

```
      subroutine force(rcut2)
          ・ ・ ・ ・ ・ ・
      common/winter/ itab(400,n)
      parameter(npe=16)
!xocl processor pe(npe)
!xocl subprocessor pes(npe)=pe(1:npe)
!xocl index partition ip=(pes, index=1:n, part=cyclic)
!xocl local itab(:,/ip)
          ・ ・ ・ ・ ・ ・
!xocl spread do/ip
      do  270  i=1,ipmax
          ・ ・ ・ ・ ・ ・
```

```
*vocl loop,novrec
        do  271  k=1,icol(i)
          ・・・・・・
  271   continue
          ・・・・・・
  270   continue
!xocl end spread sum(fx),sum(fy)
          ・・・・・・
```

Fig. 5.7 Parallel programming for force

(c)table の並列化

table は粒子の位置を入力し，自分の遮蔽距離以内にいる粒子番号の表を出力する．計算は２重ループで，2.5 節で示した並列性から粒子毎の計算は並列に行うことができる．この表は力の計算のみに用いられ並列計算以外では使用されないため，他のプロセッサが担当する部分の値を持つ必要がない．従って，表 itab の unify 転送を行う必要はないため，並列オーバーヘッドは殆ど発生しない．α r_{cut} 内にある粒子数 icol の最大を調べるため，プ!xocl end spread max(imax)によってプロセッサに跨がって icol の最大値を調べることのみが，並列オーバーヘッドとなる．この２重ループは，内側の do ループの回転数が外側の do ループのインデックスに依存するため，ロードバランスの考慮をする必要があり，これを cyclic 手続分割で行う．

並列化プログラムを Fig.5.8 に示す．注意すべきことは，itab は force で使用するため，index partition で定義した ip と同じものを force で使用することである．

```
        subroutine table(iclock,rint2)
        ・・・・・・
        common/winter/ itab(400,n)
        parameter(npe=16)
!xocl processor pe(npe)
!xocl subprocessor pes(npe)=pe(1:npe)
!xocl index partition ip=(pes, index=1:n, part=cyclic)
!xocl local itab(:,/ip)
        ・・・・・・
!xocl spread do/ip
      do 101 i=1,ipmax
        ・・・・・・
        do 100 j=i+1,n
        ・・・・・・
        if(imax.lt.icol(i)) imax=icol(i)
        ・・・・・・
100   continue
      ・・・・・・
101   continue
!xocl end spread max(imax)
        ・・・・・・
```

Fig. 5.8 Parallel programming for table

(d)propcel の並列化

propcel は入力を粒子の位置，運動量，エネルギーとし，各物理量の密度を出力する．計算は 1 重ループで，2.5 節で示した並列性から粒子毎の計算は並列に行うことができる．物理量を求めるため系を 40×20 のセルに分割し，各セル毎に物理量を積算し密度とする．この積算計算は回帰計算のためベクトル化できない．従って，force や pcross のようにベクトル化をして時間コストを減らすことはできない．一方，積算計算がプロセッサ間の総和計算 sum に集約されるため，並列化するとができる．Fig.5.9 にこれを示す．この並列化で並列性能が期待できる理由は，do ループの回転数が n(=7200)に対し，プロセッサ間の総和計算はセル数(=800)と計算回数のオーダーが異なるためである．

```
      subroutine propcel(isampa)
            ・・・・・・
      parameter(npe=16)
!xocl processor pe(npe)
!xocl subprocessor pes(npe)=pe(1:npe)
!xocl index partition ip1=(pes, index=1:n, part=band)
!xocl index partition ip2=(pes, index=1:npe, part=band)
     ・・・・・・
      do 309 j=1,ny
      do 308 i=1,nx
      ekinc(i,j) = 0.0
      pmcx(i,j) = 0.0
      pmcy(i,j) = 0.0
      npc(i,j) =0
 308  continue
 309  continue
      ・・・・・・
!xocl spread do/ip1
      do 303 j=1,n
      nxw = nxc(j)
      nyw = nyc(j)
      npc(nxw,nyw)=npc(nxw,nyw)+1
      ekinc(nxw,nyw)=ekinc(nxw,nyw)+(vx(j) * vx(j) + vy(j) * vy(j))
      pmcx(nxw,nyw)=pmcx(nxw,nyw)+vx(j)
      pmcy(nxw,nyw)=pmcy(nxw,nyw)+vy(j)
  303 continue
!xocl end spread sum(npc), sum(ekinc), sum(pmcx), sum(pmcy)
      ・・・・・・
```

Fig. 5.9　Parallelized propcel including redundant communication.

5.4 並列化(フェーズ 2)

この MD コードで扱う代表的配列は 3 種類ある．一つは粒子に関連した配列でその大きさは 7200 要素で合計 11 個ある．次にセルに関連した配列で大きさは 40×20 要素で、8 個である．最後にブックキーピング法のリストの配列 itab で，大きさは 7200×400 である．

そこで，要素数が他よりも１桁大きいitabをデータ分割した．itabはforceのみで用いられるため，容易にデータ分割できる． itabのデータ分割方法をFig.5.8に示す．!xocl local itab(:,/ip)は，２次元目をforceで使用するのと同じipでデータ分割することを示す． このデータ分割の結果， 各プロセッサで必要とするitabのメモリは1 / Npに削減できる．

この分割でのプログラミング上の留意点は，local文を用いることである．これは，global文を使用すると通信が発生するためである．local 文を用いると，他のプロセッサの itabの値を参照することはできないが，この場合ipはforceの手続き分割と同じであり，同一プロセッサのitab値の参照のみで計算ができる．

5.5 不要な通信の削除

5.3節(d)で示したpropcelの並列化では，Fig.5.9のdo 303においてプロセッサ間の総和を配列ekinc, pmcx, pmcy, npcに対して取ったため，通信によるオーバーヘッドで殆ど並列性能が向上しない．これは，このdoループが粒子数nに比例した計算量のためである．そこで，プロセッサ間の総和計算の回数を減らし，不要な通信を削除するアルゴリズムを考案した．これは，式(2.9)で示したように，物理量の計算が時間積分の平均値で表わせることにより可能となる．即ち， ekinc, pmcx, pmcyは時刻tに正確に計算できれば良いので，その計算を行うdo306でプロセッサ間の総和をとることにする．一方，各セル毎の密度平均が必要なため，npcは各時間ステップ毎にプロセッサ間の総和を計算する．

このアルゴリズムをFig.5.10に示す．do303で配列npc のみに対するプロセッサ間の総和を行い，残りはdo 306で総和を取る．この方法により，現在コードが採用しているサンプリング時間間隔2000ステップでは， ekinc, pmcx, pmcy のプロセッサ間の総和計算を1/2000に削減できるため，Fig.5.9 do303のプロセッサ間の総和計算時間をほぼ75%(=3/4)削減できる．

```
      subroutine propcel(isampa)
       ・・・・・・
      parameter(npe=16)
!xocl processor pe(npe)
!xocl subprocessor pes(npe)=pe(1:npe)
!xocl index partition ip1=(pes, index=1:n, part=band)
!xocl index partition ip2=(pes, index=1:npe, part=band)
       ・・・・・・
      do 309 j=1,ny
      do 308 i=1,nx
      ekinc(i,j) = 0.0
      pmcx(i,j) = 0.0
      pmcy(i,j) = 0.0
      npc(i,j) =0
 308  continue
 309  continue
       ・・・・・・
c accumulate physical parameters into each cell.
```

```
!xocl spread do/ip1
      do 303 j=1,n
      nxw = nxc(j)
      nyw = nyc(j)
      npc(nxw,nyw)=npc(nxw,nyw)+1
      ekinc(nxw,nyw)=ekinc(nxw,nyw)+(vx(j) * vx(j) + vy(j) * vy(j))
      pmcx(nxw,nyw)=pmcx(nxw,nyw)+vx(j)
      pmcy(nxw,nyw)=pmcy(nxw,nyw)+vy(j)
  303 continue
!xocl end spread sum(npc)
c compute density of physical parameters on each cell.
      do 307 j=1,ny
      do 304 i=1,nx
       if(npc(i,j).eq.0) then
        ekinc(i,j)=0.0d0
        pmcx(i,j)=0.0d0
        pmcy(i,j)=0.0d0
       else
        ekinc(i,j)=ekinc(i,j)/npc(i,j)
        pmcx(i,j)=pmcx(i,j)/npc(i,j)
        pmcy(i,j)=pmcy(i,j)/npc(i,j)
       end if
  304 continue
  307 continue

      do 305 j=1,ny
      do 305 i=1,nx
        aekinc(i,j)=aekinc(i,j)+ekinc(i,j)
        apmcx(i,j)=apmcx(i,j)+pmcx(i,j)
        apmcy(i,j)=apmcy(i,j)+pmcy(i,j)
 305  continue
c check sampling time and compute physical parameters.
      if( isampa.eq.0 ) then
      do 9309 j=1,ny
      do 9308 i=1,nx
      ekinc(i,j) = 0.0
      pmcx(i,j) = 0.0
      pmcy(i,j) = 0.0
      npc(i,j) =0
9308  continue
9309  continue
!xocl spread do/ip2
        do 306 k=1,npe
        do 306 j=1,ny
        do 306 i=1,nx
         vxx=apmcx(i,j)/isampc
         apmcx(i,j)=sbt*vxx
         pmcx (i,j)=pmcx(i,j)+apmcx(i,j)
         vyy=apmcy(i,j)/isampc
         apmcy(i,j)=sbt*vyy
         pmcy (i,j)=pmcy(i,j)+apmcy(i,j)
         aekinc(i,j)=sbt**2/2./rtc*(aekinc(i,j)/isampc-vxx**2-vyy**2)
         ekinc (i,j)=ekinc(i,j)+aekinc(i,j)
 306   continue
```

```
!xocl end spread sum(pmcx),sum(pmcy),sum(ekinc)
          write(31,2020)   ((i,j,ekinc(i,j),pmcx(i,j),pmcy(i,j),i=1,nx),j=1,ny)
 2020   format(1h ,'*i,j,t,vx,vy = ',2i4,3e15.6)
            . . . . . .
       end if
            . . . . . .
```

Fig.5.10 Removal of redundant communication for the parallelized propcel

6. Paragonにおける並列化

Paragonは，倍精度の浮動小数点演算で75MflopsのRISCプロセッサをノードとし，それらを送受信各々200MB/secのメッシュネットワークで接続した並列計算機である．プロセッサは，浮動小数点計算に対してソフトウエアパイプライニングができる．パイプラインの段数は加算，乗算，ロードに対して，単精度では3，倍精度では乗算2，加算とロードは3である［12］．Paragonコンパイラはソフトウエアパイプライニングのために計算順序を変える機能を持つ．またベクトル化と呼ばれる機能を持つ．これは，多重doループを対象にしたソフトウエアパイプライニングのための最適化，キャッシュ利用の最適化，内積計算等をベクトル計算できるルーチンに置き換える3つのプロセスから成り，doループのくり返し計算に対して有効である．キャッシュは，命令とデータに各々16KBを持つ．この内データキャッシュ16KBの一部はベクトル計算で使用される．その大きさはコンパイルオプションにより変えることができ，デフォールト値は4096Bである［13］．

本論文で使用するParagonシステムは，メモリ量が32MBのノードからなる256ノードのシステムである．また，MDコードは全ルーチン倍精度計算である．

6.1 プログラム分析

オリジナルプログラムの1プロセッサの時間コスト分布をTable6.1に示す．この値は，経過時間測定関数dclockをサブルーチンの先頭と終わりに挿入して測定した．コンパイルは if77 -nx -Mr8 -Mr8intrinsics *.f で行った．単一プロセッサの性能が最大75Mflopsのため，時間ステップ数は2000回から50回に短縮して測定した．表は，主要コストルーチンが，force, propcel, table, pcorssであることを示す．また表中には現れないが，chkbndの時間はpcross及びpmovesに含まれ，殆どの時間をpcrossで費やす．

Table 6.1 Time cost distribution of original MD

Subroutine Name	Time Cost (sec)	Time Cost (%)
force	363.6	74.8
table	48.9	10.1
pcross	33.8	7.0
propcel	28.9	5.9
pmoves	4.0	0.8
(others)	(7.0)	(1.4)
TOTAL	486.2	100.0

6.2 単一プロセッサにおける最適化

Paragonは，コンパイルオプションとコンパイルディレクティブによって，ソフトウエアパイプライニング，インライン展開，ベクトル計算，割算の高速割算（精度は変わる）を行うことができる［13］．そこで最適化のためのコンパイルオプションを変え，その効果を調べた．その結果，単一プロセッサの実行では「-nx -Mr8 -Mr8intrinsics -O4 -Knoieee」，また並列実行では，利用したプロセッサ間の総和計算を行うサブルーチンtgdsum[14]のベクトル化のため，「-nx -Mr8 -Mr8intrinsics -O4 -Minfo=loop -Mvect -Knoieee」を適用した．

ここに，-O4はソフトウエアパイプライニングとグローバルな最適化を，-Mvectはベクトル化を，-Minline=50は50ステップ以下のルーチンをインライン展開の対象にすることを，意味する． また，-Mr8 -Mr8intrinsicsは倍精度計算をするためのオプションである．-nxはプログラムを計算パーティションで計算させるためのオプションで必須である．

(a) -Knoieee オプションによる最適化

-Knoieeeは，除算においてIEEEで保証する計算結果とは異なるが，高速なライブラリを適用する．除算，根の計算等が高速化される．

(b) -Mvectオプションによるベクトル化

ベクトル化されたか否かは，コンパイルオプション-Minfo=loopで知ることができる．

-Mvect を指定し，forceの do271 (Fig.5.2a参照)の直前に回帰計算でないことを明記するコンパイラディレクティブ cdir$l nodepchk を挿入したが，ベクトル化されなかった．do271中の if文をコメントアウトするとベクトル化することから，原因はif文の存在によるものと推測できる．tableもforceと同様にベクトル化されない．pcross, pmovesは，doループ中で呼ばれているchkbndが コンパイルオプション-Minline=50でインライン展開されるが，ベクトル化されない．またこれら全てのルーチンの主要時間コストdoループが，ソフトウエアパイプライニングされない．

並列実行時に用いるプロセッサ間の総和を取るユーザルーチン tgdsum は，ベクトル化により性能が向上する．

(c)ブロックアクセスによるキャッシュの最適化

ベクトル計算と同様にitabのインデクスを逆転させた（Fig.5.2b参照）． これは，オリジナルプログラムのアクセスの間隔が32kB(=400要素×8Byte)で，キャッシュのミスヒットが予想されるためである．尚，サブルーチンitabのアクセスは，Fig.5.3bと同様にforceの変更に従って逆にし最適化した．

6.3 並列化(フェーズ 1)

5.3 節で示した並列化アルゴリズムを，Paragon のメッセージパッシングライブラリで実現した．この並列化方法は，プロセッサ間総和計算のスピードがキーポイントとなるため，Paragon のライブラリ gdsum より高速なユーザサブルーチン tgdsum[14]を用いた．このルーチンのベクトル化のため， コンパイルオプション-Mvect を用いた．

(a)プログラム分析

Table6.2 は単一プロセッサの最適化を実施した後のコスト分布である．並列化するサブルーチンは，force, propcel, table であるので，表より並列化率が 92.2%になることが分かる．この並列化率を式(4.3)に代入し，並列効率 Ep を仮定してそれを満たすプロセッサ数 Np を求めると，(Ep=0.1,Np=116)，(Ep=0.5,Np=14)，(Ep=0.8,Np=4)，(Ep=0.9,Np=2)となる．ここで使用した，日本原子力研究所の Paragon システムのプロセッサ数は 256 台，この内 1 ユーザが常時使えるプロセッサ数は 128 台である．従って，式(4.1)より最大で約 12 倍の並列計算を期待できる．

Table6.2 Time cost distribution of optimized MD

Subroutine Name	Time Cost (sec)	Time Cost (%)
force	149.5	68.6
table	40.5	18.6
propcel	10.9	5.0
pcross	9.8	4.5
pmoves	3.7	1.7
(others)	(3.5)	(1.6)
TOTAL	217.9	100.0

(b)force の並列化

並列化は，Fig.6.1 の様に行った．手続き分割は，ストライド nnode を一定にした do ループで実現する．mnode は自プロセッサの番号である．また nnode はプロセッサの総数である．計算した fx, fy のプロセッサ間の総和は，サブルーチン tgdsum により行う．tgdsum は通信時のデータの衝突を避ける工夫をした総和アルゴリズムを用いたユーザサブルーチンである．

```
program main
  ・・・
call tgdinit()
nnode=numnodes()
```

```
      mnode=mynode()
       ・ ・ ・
      end

      subroutine force(rcut2)
      dimension tmp(n)
       ・ ・ ・
      do  270  i=mnode+1,ipmax,nnode
        iii=iii+1
        xi = x(i)
        yi = y(i)
      do  271  k=1,icol(iii)
        j    = itab(k,iii)
       ・ ・ ・
  271  continue
  270  continue
       call tgdsum(fx,n,tmp)
       call tgdsum(fy,n,tmp)
       ・ ・ ・
```

Fig.6.1 Parallel programming for force

(c)table の並列化

並列化は， force と同様に Fig.6.2 の様に行った．プロセッサに跨がる icol の最大値は，このサブルーチンが呼び出し回数が少ないため，標準のシステムコール gdhigh を用いた．

```
       ・ ・ ・
      iii=0
      do 101 i=mnode+1,ipmax,nnode
        iii=iii+1
      xi = x(i)
      yi = y(i)
      icol(iii) = 0
c
      do 100 j=i+1,n
       rx = (x(j)-xi)**2+(y(j)-yi)**2
       if(rx.lt.rint2) then
         icol(iii) = icol(iii)+1
         itab(icol(iii),iii) = j
       end if
  100  continue
      if(imax.lt.icol(iii)) imax=icol(iii)
  101  continue
      rimax=imax
      call gdhigh(rmax,1,wrmax)
      imax=rimax
      ・ ・ ・
```

Fig.6.2 Parallel programming for table

(d)propcel の並列化

並列化は，Fig.6.3 のように行った．

```
      subroutine propcel(isampa)
        ・・・
      do 303 j=mnode+1,n,nnode
        nxw = nxc(j)
        nyw = nyc(j)
        npc(nxw,nyw)=npc(nxw,nyw)+1
        ekinc(nxw,nyw)=ekinc(nxw,nyw)+(vx(j) * vx(j) + vy(j) * vy(j))
        pmcx(nxw,nyw)=pmcx(nxw,nyw)+vx(j)
        pmcy(nxw,nyw)=pmcy(nxw,nyw)+vy(j)
  303 continue
      call tgdsum(npc,nx*ny,tmp)
      call tgdsum(ekinc,nx*ny,tmp)
      call tgdsum(pmcx,nx*ny,tmp)
      call tgdsum(pmcy,nx*ny,tmp)

        ・・・
```

Fig.6.3　Parallelized propcel including redundant communication.

6.4 並列化(フェーズ 2)

5.4 節と同様の考え方で，配列 itab をデータ分割した．

itab のデータ分割方法を Fig.6.4 に示す．プロセッサ数 npe を parameter 文で与えることにより，データ分割を行うことができる．尚，この方法ではプロセッサ数 npe をコンパイル時に与えるため，プロセッサ数を変更する時は，再コンパイルが必要である．

```
parameter(npe=128)
parameter(ndiv=n/npe+1)
common/inter/icol(n),itab(400, ndiv)
```

Fig. 6.4 Data decomposition of itab

6.5 不要な通信の削除

Fig.6.3 の do303 の直後に行われる ekinc，pmcx，pmcy，npc のプロセッサ間の総和は，5.5 節で示した様に，ekinc，pmcx，pmcy について削除できる． Fig.6.4 にこれを示す．即ち，do303 直後の ekinc，pmcx，pmcy に対するプロセッサ間の総和計算を do306 の直後で行う．注意すべきことは， npc を実数と宣言することである．Paragon におけるソフトウエアパイプライニング，ベクトル化は，浮動小数点計算みに対して行われるためである．

```
      subroutine propcel(isampa)
        . . .
      do 303 j=mnode+1,n,nnode
        nxw = nxc(j)
        nyw = nyc(j)
        npc(nxw,nyw)=npc(nxw,nyw)+1
        ekinc(nxw,nyw)=ekinc(nxw,nyw)+(vx(j) * vx(j) + vy(j) * vy(j))
        pmcx(nxw,nyw)=pmcx(nxw,nyw)+vx(j)
        pmcy(nxw,nyw)=pmcy(nxw,nyw)+vy(j)
  303 continue
      call tgdsum(npc,nx*ny,tmp)
c
      do 307 j=1,ny
      do 304 i=1,nx
        if(npc(i,j).eq.0) then
        ekinc(i,j)=0.0d0
        pmcx(i,j)=0.0d0
        pmcy(i,j)=0.0d0
      else
        ekinc(i,j)=ekinc(i,j)/npc(i,j)
        pmcx(i,j)=pmcx(i,j)/npc(i,j)
        pmcy(i,j)=pmcy(i,j)/npc(i,j)
      end if
  304 continue
  307 continue
        . . .
      if( isampa.eq.0 ) then
      do 9309 j=1,ny
      do 9308 i=1,nx
        ekinc(i,j) = 0.0
        pmcx(i,j) = 0.0
        pmcy(i,j) = 0.0
        npc(i,j) =0
 9308  continue
 9309  continue
      do 306 j=1,ny
      do 306 i=1,nx
        vxx=apmcx(i,j)/isampc
        apmcx(i,j)=sbt*vxx
        pmcx (i,j)=pmcx(i,j)+apmcx(i,j)
        vyy=apmcy(i,j)/isampc
        apmcy(i,j)=sbt*vyy
        pmcy (i,j)=pmcy(i,j)+apmcy(i,j)
        aekinc(i,j)=sbt**2/2./rtc*(aekinc(i,j)/isampc-vxx**2-vyy**2)
        ekinc (i,j)=ekinc(i,j)+aekinc(i,j)
  306   continue
      call tgdsum(pmcx,nx*ny,tmp)
      call tgdsum(pmcy,nx*ny,tmp)
      call tgdsum(ekinc,nx*ny,tmp)
      if(mnode.eq.0) then
        write(31,2020)   ((i,j,ekinc(i,j),pmcx(i,j),pmcy(i,j),i=1,nx),j=1,ny)
      end if
        . . .
```

Fig.6.3 Removal of redundant communication for the parallelized propcel

7.並列化とその性能

VPP500 と Paragon の性能測定の結果，計算時間短縮のためには，単一プロセッサにおける性能向上が重要であることがわかった．2.3 節で与えられた計算パラメータでは，両計算機で最大 10 倍弱の並列性能を得た．また遮蔽距離 r_{cut} と粒子数 n を変えパラメータサーベイを行った結果，r_{cut} を大きくしていくとプロセッサ数 100 台規模でもスケーラビリティが得られる場合があることがわかった．このことは，r_{cut} が大きくなるポテンシャル形状で計算した場合，5 章と 6 章で述べた「do ループレベルの並列化」でプロセッサ数 100 の規模で十分な並列性能が得られる可能性があることを意味する．また r_{cut} によっては，5 章と 6 章で述べた並列化のさらに一部の並列化で十分な性能が得られることがわかった．これは，主要時間コスト do ループを 1 つ 1 つ段階的に並列化することによって，並列化作業時間を減らせる場合がある．

7.1 単一プロセッサにおける性能向上

2.3 節で与えられた計算パラメータで性能を測定した．VPP500 においてはベクトル化が，Paragon においてはキャッシュの有効利用が性能に大きく影響を与えた．

(a) VPP500 における最適化の効果

経過時間を測定するサブルーチン gettod を使用して最適化の効果を測定し，Table7.1 に示す．主にベクトル化によって，全体の性能が 3.3 倍向上した．表はまた，ベクトル化によって時間コストが一部に集中したことを示す．ベクトル化により pcross，pmoves，chkbnd が高速化され，その結果並列化対象サブルーチン force，table，propcel の時間コストの占める割合が増し，並列化率が向上した．Table7.1 の最適化されたコードのコスト分布より，上位 3 ルーチンの全体に対する並列化率は 98.1%，式(4.1)にこの値を代入してプロセッサ数を無限にすると，並列性能の最大値は 53 倍である．表中の Code TOTAL を時間ステップ数 2000 とサンプリング数 6 で割ると，1 タイムステップ当たりの計算時間は，0.0584 秒となる．

Table7.1 Uni-processing performance on VPP500. Compile option -Oe used in original code, which is exchanged to -Oe -Ne,50 in optimized one.

Sub. name	force	table	propcel	pcross	pmoves	Code TOTAL
original (sec)	1635.1	223.4	32.8	167.7	60.1	2123.5
opt. (sec)	633.5	20.6	33.5	4.8	1.6	701.0
speedup	2.6	10.8	1.0	34.9	37.6	3.3

(b) Paragon における最適化の効果

最適化の効果を Table7.2 にまとめる．主にコンパイルオプション-Knoieee により，全体の性能が 1.5 倍向上した．このオプションにより高速化されたサブルーチンは，force，propcel，pcross，及び pcross から呼ばれている chkbnd である． force において，配列 itab のアクセスを連続して行ったキャッシュの最適化によりさらに性能が 1.5 倍向上し，結果として 2.2 倍向上した．その結果，対象 3 ルーチンの全体に対する並列化率は 92.2%，式(4.1)にこの値を代入してプロセッサ数を無限にすると，並列性能の最大値は 13 倍である．表中の Code TOTAL を時間ステップ数 50 とサンプリング数 6 で割ると，1 タイムステップ当たりの計算時間は， 0.729 秒となる．

Table7.2 Uni-processing performance on Paragon. Compile option -O2 used in original code, which is exchanged to -O4 -knoieee in optimized one.

Sub. name	force	table	propcel	pcross	pmoves	Code TOTAL
Original (sec)	363.6	48.9	28.9	33.8	4.0	486.2
opt. (sec) (by compiler)	261.4	41.1	10.8	9.1	4.0	330.2
Relative speedup	1.4	1.2	2.7	3.7	1.0	1.5
opt. (sec) (by cache)	149.5	—	—	—	—	217.9
Relative speedup	1.7	—	—	—	—	1.5
Total speedup	2.4	1.2	2.7	3.7	1.0	2.2

7.2 並列性能

(a) VPP500 における並列化の効果

並列性能を Fig.7.1 に示す．最適化された単一プロセッサの性能に対し，プロセッサ数 16 で 9 倍となる． この並列効率は 58%である．従って，プロセッサ数 10 前後で，システムを効率良く利用することができる．理想曲線からの差異の原因と考えられる並列化率，通信，ロードバランスの乱れを個別に調査した結果，その原因は並列化率と通信であることがわかった．Table7.3 よりその割合は，プロセッサ数 16 において，逐次実行部の計算時間に占める割合が 21%，プロセッサ間の総和計算時間が 15%であった．

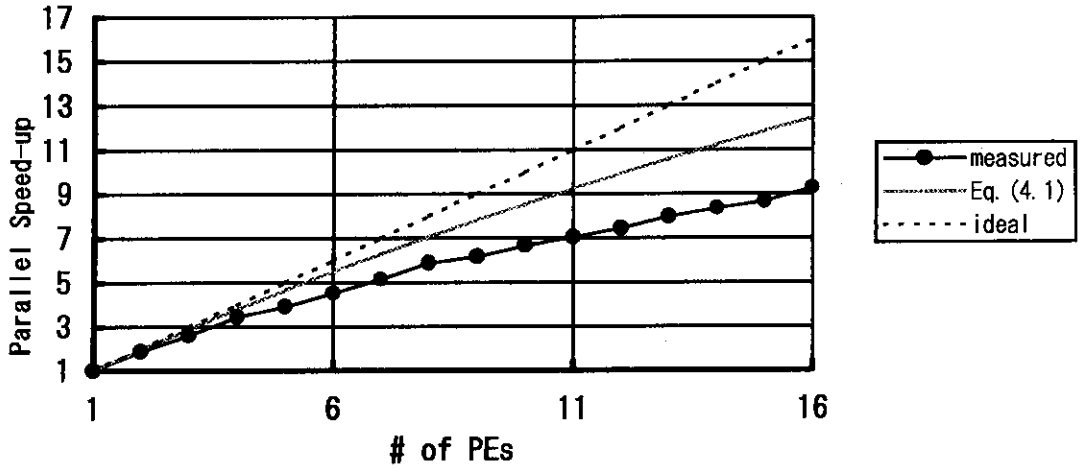

Fig.7.1 Parallel speed-up of do-loop level of parallel programming on VPP500.

i)並列化率

式(4.1)に並列化率 98.1%，プロセッサ数を代入して並列性能を計算すると，並列化率による並列性能の上限がわかり， Fig.7.1 に示す通りプロセッサ数 16 で 12.5 倍であり，並列効率は 78%である．この並列化率は pcross と pmoves の逐次実行が原因の一つであり，Table7.3 からこの部分の計算時間は，プロセッサ数 16 で 6.3 秒であることがわかる．しかし表はそれ以外にもプログラムに広く分散して 9.7 秒の逐次実行部分があり，合計 16.0 秒が逐次実行時間であることを示す． これを割合で表わすと全体の 21%である． Fig.7.1 の実測値はまた，並列性能が並列化率以外の原因で低下していることを示す．

ii)通信

Table7.3 に， プロセッサ数の増加に伴う主要時間コストルーチンの時間と，それに含まれる通信を伴う関数，総和(sum)と最大(max)の時間を示す．表より， force と propcel のプロセッサ間の総和計算時間がプロセッサ数の増加と共に増加し，プロセッサ数 16 では 11.1 秒で，全体の約 15%になることがわかる．VPP500 では，この時間はほぼ通信時間である[8]．表よりプロセッサ数 16 の時のサブルーチン毎の並列効率を計算すると，force は 73%，table は 86%，propcel は 50%である． sum， max の時間を差し引いて並列効率を計算すると，各々89%，87%，84%である．このことは， force と propcel において，通信により並列効率が下がっていることを示し，特に propcel において通信オーバーヘッドの影響が大きいことを示す．

Table7.3 Time cost distribution and parallel overhead using communication on VPP500.

(sec)

# of PEs	1	4	8	16
force	633.5	173.5	95.7	54.0
(sum)	(-)	(7.5)	(8.3)	(9.4)
table	20.6	5.6	2.8	1.5
(max)	(-)	(0.01)	(0.02)	(0.02)
propcel	33.5	11.0	7.2	4.2
(sum)	(-)	(1.3)	(1.3)	(1.7)
pcross	4.8	4.7	4.8	4.7
pmoves	1.6	1.6	1.6	1.6
Code TOTAL	700.9	204.2	119.6	75.7
(ovhd)	(-)	(8.8)	(9.6)	(11.1)
Parallel efficiency	1	0.858	0.733	0.579

iii)ロードバランスの乱れにより発生する待ち時間の測定

並列化したforceのdoループに対し各プロセッサ毎の計算時間を積算し，プロセッサ毎の計算時間を平均し，残差の最大値を%で表わすことによりロードバランスの乱れを評価した． その結果を Table7.4 に示す．この表によればロードバランスの乱れは，プロセッサ数16で最大で0.62%であり，並列オーバーヘッドに寄与していないことがわかる．

Table7.4 Load balance of force on VPP500

# of PEs	4	8	16
Deviation (%)	0.46	0.45	0.62

(b) Paragonにおける並列化の効果

並列性能を Fig.7.2 に示す．最適化された単一プロセッサの性能に対し，プロセッサ数32で5.2倍となり，それ以上プロセッサ数を増やしても並列性能は向上しない． この並列効率は16%である．従って，Paragonシステム上では，並列効率を高くしてシステムを利用することはできない．理想曲線からの差異の原因である並列化率，通信を含んだ関数の処理時間を個別に調査した結果，その第一の原因は並列化率であることがわかった．その割合は，プロセッサ数32において，逐次実行の計算時間に占める割合が41%，プロセッサ間の総和計算時間が25%であった．

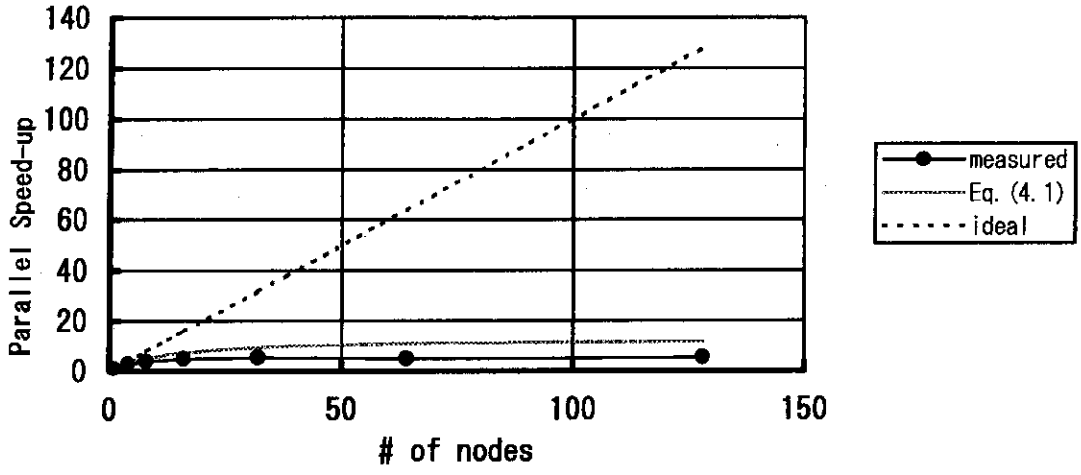

Fig.7.2 Parallel speed-up of do-loop level of parall programming on Paragon.

i)並列化率

式(4.1)に並列化率 92.2%，プロセッサ数を代入して並列性能を計算すると，並列化率による並列性能の上限がわかり， Fig.7. 2 に示す通りプロセッサ数 32 で. 9.4 倍であり，並列効率は 0.29 である．この並列化率は主に pcross と pmoves が逐次実行のためである．Table7.5 からこの部分の計算時間は，プロセッサ数 32 で 13.5 秒であることがわかる．表はそれ以外にもプログラムに分散して 3.6 秒の逐次実行部分があることを示す．合計 17.1 秒が逐次実行時間であり，これを割合で表わすと全体の 41%である． Fig.7.2 は，並列性能が並列化率が原因で向上しないことを示す．

ii)通信を含んだ関数の処理時間

Table7.5 に，プロセッサ数の増加に伴う主要時間コストルーチンの時間と，それに含まれる通信を伴う関数の時間を示す．表より，プロセッサ間の総和計算時間 tgdsum がプロセッサ数の増加と共に増加し，プロセッサ数 4 では 4.2 秒であるのに対し，プロセッサ数 128 では 14.4 秒である．これは， tgdsum で使用しているアルゴリズムの通信と総和計算時間が $\log_2 Np$ に比例する[8]ためである．ここに Np はプロセッサ数である．この式に Np=4，128 を代入しその比を取ると 3.5，実測値のそれは 3.4 であることがこれを裏付ける．表中の propcel の並列効率が他のサブルーチンに比べプロセッサ数の増加に対して著しく低下するのは，通信と総和計算時間が $\log_2 Np$ に比例して増加するためである．この通信と総和計算時間がいかに大きいかは，この時間を無視して並列計算時間を予測することによりわかる．式(4.1)において Rp=1 とし， Np を与えて表のプロセッサ 1 の時の計算時間 217.4 秒を用いると，予測値は Np=32 で 6.8 秒， Np=16 で 13.6 秒である．各々の Np に対するとプロセッサ間の総和計算時間は，各々10.4 秒，8.4 秒である． Np=32 でプ

ロセッサ間の総和計算時間が他の計算部分を上回ることがわかる．

この最大並列性能 5.2 がどの程度のものかは，この値に単一プロセッサの最大性能 75Mflops を掛けるとわかり易い．その性能値は，単一プロセッサのベクトル計算機に近づく．無論，6.2 節で示したように主要ルーチンはベクトル化されておらず実効性能はこの数十パーセント程度と予想されるが，計算機利用者，並列化作業者にとっての目安となる．

一方，同様な並列化方法でも，他のポテンシャルを使用する場合には，並列効率向上と高い絶対性能を出せる可能性がある．これは 7.2.3 節に示す．

Table7.5 Time cost distribution and parallel overhead using communication on Paragon.

(sec)

# of nodes	1	4	8	16	32	64	128
force	149.5	44.0	27.4	20.0	17.1	20.1	17.0
table	40.5	10.4	5.1	2.6	1.3	0.7	0.6
propcel	10.4	7.2	7.0	6.2	6.3	6.8	6.5
pcross	9.8	9.8	9.8	9.8	9.8	9.8	9.8
pmoves	3.7	3.7	3.7	3.7	3.7	3.7	3.7
others	3.6	4.5	5.6	3.5	3.6	4.0	3.5
Code TOTAL	217.4	79.6	58.6	45.8	41.8	44.8	41.1
(tgdsum)	(-)	(4.2)	(6.2)	(8.4)	(10.4)	(16.1)	(14.4)
Parallel efficiency	1	0.683	0.464	0.297	0.163	0.076	0.041

(c) パラメータサーベイ

2 章で述べたポテンシャルの遮蔽距離 r_{cut} は，3 σ であった．この r_{cut} は，パラメータ $(m)_{LG}$，$(n)_{LG}$ が変わる時，また異なったポテンシャルを用いた時，変化する． r_{cut} が大きくなり，計算している系の大きさに近づくに従って force の計算時間の占める割合が増し，並列化率が向上する．一方，プロセッサ間の総和計算量は粒子数 n が変わらず同じであるため，プロセッサ間の総和計算時間の占める割合が相対的に減少する．これらの相乗効果で，r_{cut} が大きくなった時，プロセッサ数を増した時のスケーラビリティが期待できる．そこでその効果を調べた． ここでは簡単化のため，ポテンシャルパラメータや形状を変えそれに適した r_{cut} を用いる変わりに， r_{cut} のみを変えてパラメータサーベイを行った．この方法では定量的な評価はできないが，ポテンシャルの種類を問わず，計算には粒子間の距離の自乗の計算が必要となり，これがポテンシャル計算の時間コストとして無視できないことから，定性的な予測が可能である． 尚， $r_{cut}=\infty$ の計算は，Fig.5.2b 及び Fig.6.1 の force の do271 を j=i+1,n に変更して実行した結果を使用した．

i)VPP500

r_{cut} を変化させたパラメータサーベイの結果を Fig.7.4a に示す． r_{cut} が大きくなるに従って，並列効率が大きく向上する．このことから， r_{cut} を大きく取れるポテンシャルを使用した場合，５章で示した並列化方法のみで，数十台のレンジで実用的な並列効率が得られる．

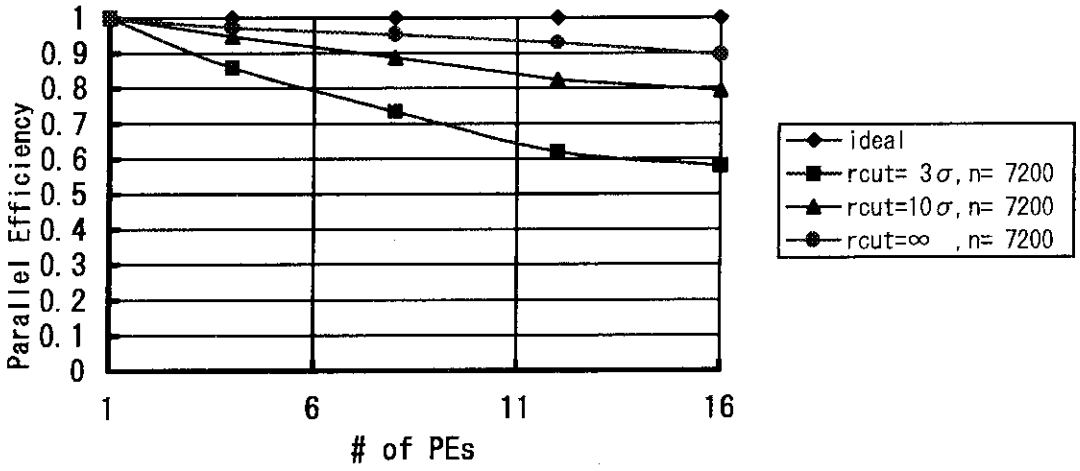

Fig.7.4a Parallel efficiency under varying r_{cut} on VPP500.

また Fig.7.4b に示す様に，n の増加に対して並列効率は緩やかに向上する．この MD コードでは，n の増加に対してスケーラビリティ向上が期待できる．しかし n の増加と共に計算量と通信量の両方が増え，並列効率の増減は単一プロセッサの計算性能と通信性能のバランスにより成り立っているため，異なるポテンシャルで実行した場合，反対に減少する場合があることも予測できる．

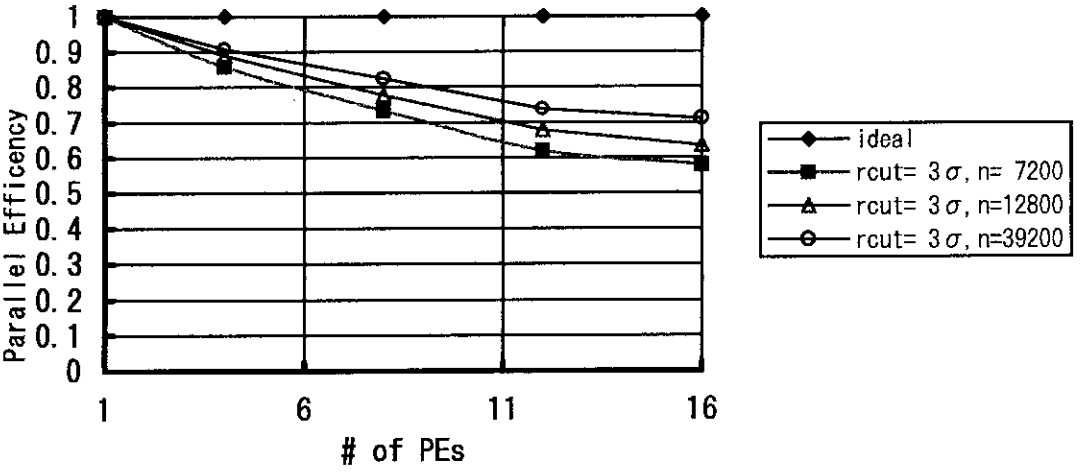

Fig.7.4b Parallel efficiency under varying n on VPP500.

ii)Paragon

パラメータサーベイの結果を Fig.7.5 に示す．r_{cut} が無限大では 128 台で約 50%の並列効率を得ることができる．このことから，r_{cut} を大きく取れる長距離力等のポテンシャルを使用した場合，6 章で示した並列化方法のみで，数十台から百台のレンジでスケーラビリティが得られることを予測できる．

一方，n の増加に対して並列効率は変化しない．n の増加と共に計算量と通信量の両方が増え，単一プロセッサの計算性能と通信性能が丁度バランスしたためと考えられる．

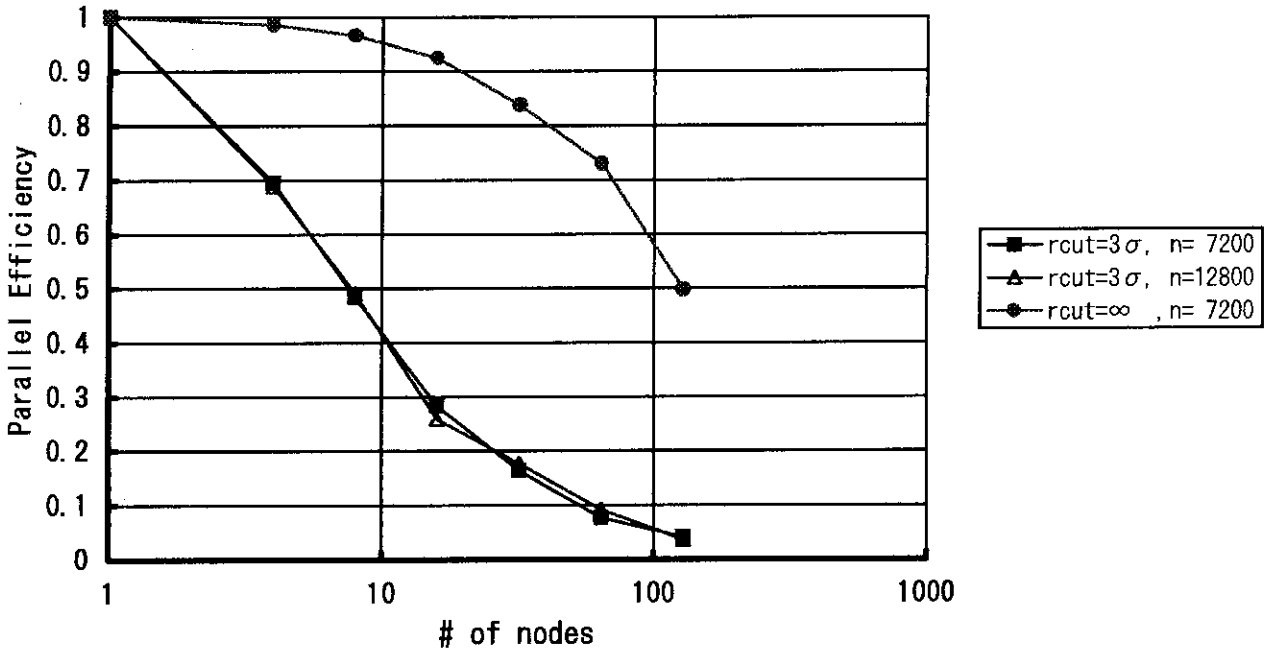

Fig.7.5 Parallel efficiency under varying computing parameters on Paragon.

7.3 並列性能と段階的並列化

MD コードの並列化は，2.3 節で示した計算パラメータに対して force，table，pcross，chkbnd，pmoves における「単一プロセッサにおける性能向上」，force，table，propcel の「do ループレベルの並列化」，propcel の総和計算のようにサブルーチン内の複数の do ループに跨った「不要な通信の削除」の段階を経て行われた．この並列化コードは 7.2 節(c)に示したように，異なった r_{cut} に対して種々の並列性能を持つ．その測定結果を注意して調べると，r_{cut} が大きい場合，propcel の並列化が必要ない場合があることがわかる．これは，force の計算時間が propcel 等のその他の計算より十分に大きくなるためである．これを Table7.6 に示す．MD コードでは，このようにより少ない作業で並列化ができる場合がある．この様に，性能測定とパラメータサーベイの結果は，上記に述べた様に段階的に並

列化を行うことが並列化作業量の減少につながることを示している．この時，実際のシミュレーションに使う計算パラメータを用いることが，ポイントである．

Table7.6 Time cost distribution for r_{cut} =10 σ on VPP500.

(sec)

# of PEs	force	table	propcel	pmoves	pcross	Code TOTAL
16	136.1	0.4	5.8	1.6	4.7	156.4

MD コードを並列計算機向けにプログラミングする研究は種々行われており[14], ここで用いた MD コードでも同様の並列化が可能である．しかし，そのためには「大幅にプログラムを書き直す作業」を必要とする．その量は，本報告書の並列化作業時間が数週間程度であるとすると，数ヶ月のオーダとなる．理由は， 並列化率を向上し通信を減らすため，各プロセッサが持つデータの依存関係をサブルーチン間に跨る広い範囲でプログラマが保証し， また，5.3 節(a)で述べたように，プロセッサ数が変化すると計算結果も変化することを防止する並列乱数発生を工夫する必要があるためである．この作業量を減らすには，２章で示した MD コードに関する知識を用いるのが早道である．しかし MD コードに関する知識を利用した時，並列化は一部の専門知識を持つ人だけが可能な特殊な作業となる．一方，段階的な並列化はこのような特殊な作業にならないで並列化が可能である．本報告書の５章と６章で示したように，２フェーズ法を用いた段階的並列化は，do ループの並列化が基本となるため，ベクトル化とほぼ同等の作業量で並列化が可能となる場合がある．

8. 議論とまとめ

既存のMDコードを2フェーズ法を用い，3つの段階「単一プロセッサ性能向上」，「doループレベルの並列化」，「不要な通信の削減」に分けて並列化した．その結果，VPP500プロセッサ数16で9倍，Paragonプロセッサ数32で5倍の並列性能を得た．その並列化コードをパラメータサーベイした結果，遮蔽距離が増加すると並列化率が向上し，かつプロセッサ間の総和計算時間に比べ力の計算時間が増えるため，プロセッサ数100の規模の計算で，並列効率が50%以上得られる場合があることがわかった．このことより，使用するポテンシャルの種類に依存して並列性能が大きく異なることがわかった．

計算時間の短縮には「単一プロセッサの性能向上」が重要である．VPP500では，ベクトル化により約300%性能が向上した．Paragonではコンパイルオプションとキャッシュの最適化により約200%性能が向上した．

ベクトル計算とスカラ計算の違いは並列化率に現れた．2.3節の計算パラメータ（粒子数7200，遮蔽距離3σ）で計算すると，VPP500では98.1%，Paragonでは92.2%である．この違いは，粒子の移動と反射を計算するサブルーチンpmoves，境界面の判定を行うpcrossとpcrossから呼ばれるchkbndが，VPP500ではベクトル化により高速計算されるため生じる．一方これらのサブルーチンは，Paragonではソフトウエアパイプライニング，ベクトル化がされなかった．doループ中にif文があるためと推測する．

また並列化率の向上で重要なことは，物理量の計算時間が無視できないことである．2.3節の計算パラメータでは，その計算を行うサブルーチンpropcelの並列化が必要であった．

今回VPP500とParagonで得られた並列化率から推測すると，既存のMDコードを並列化すると，並列化できない部分が必ず数パーセント残り，その結果並列化率は90%を少し越えるぐらいが期待できる．これは，通常期待できる並列性能は10倍のレンジであることを意味する．このため，絶対性能で高い性能を得るためには，単一プロセッサの性能が高い並列計算機を使用することが適当と考える．一方，Paragonにおける並列性能の測定結果から，並列効率を度外視した時，単一プロセッサのベクトル計算機に近い性能を得られることがわかった．更にParagonにおける測定結果は，並列性能はポテンシャルの遮蔽距離に依存し，力の計算量が粒子数の自乗に近づくに従って並列化率が向上し，100倍レンジの並列性能を期待できる場合があることを示す．

通常行われる，既存コードを大幅に書き直してアルゴリズムの変更を行って高並列率，通信の最少化，ロードバランシングを行う並列化方法では，長時間のデバッギングを必要とし，またアプリケーションと計算機に対する専用知識を必要とするため，専門知識を持つ少数プログラマのみが並列化を行うことが可能であった．これは，並列計算機を専用的に使用することと位置付けることができる．

一方ここで紹介した既存のコードの並列化を簡単に行う方法は，2フェーズ法を使用し，

「doループ単位の並列化」を段階的に行うことで，そのプログラムの計算内容を知らなくても，doループのコーディングを解析することにより並列化が可能である．これにより，一般のプログラマが容易に並列化を行うことが可能となる．これは，並列計算機を汎用的に使用することと位置付けることができる．並列計算機は並列プログラミング方法に依存してこのような２面性を持ち，今までのスーパーコンピュータ利用の延長線上で利用する場合，プログラミング方法を限定し，汎用的に利用することが必要となろう．この時，通常期待できる並列性能は10倍のレンジであろう．

謝　辞

日本原子力研究所計算科学技術推進センターの渡辺正氏には，MD コードを使用させていただいたことを感謝いたします．同じく計算科学技術推進センターの蕪木英雄氏には，MD コードの物理的内容のレクチャーをいただいたことを感謝いたします．計算科学技術推進センターの広田章氏には，一部の時間測定をしていただいたことを感謝いたします．同じく計算科学技術推進センターの南正雪氏には，使用した MD コードのプログラム構造について議論をしていただいたことを感謝いたします．

参考文献

[1]折居茂夫：第 49 回全国大会講演論文集，情報処理学会，5T-01(1994)．
[2]Matsumoto, K. and Orii, S. ： 第 47 回全国大会講演論文集，情報処理学会，2D-5(1993)．
[3]神林奨：JAERI-M 92-080，“分子動力学シミュレーションコード ISIS の開発”(1992)．
[4]渡辺正，蕪木英雄，町田昌彦：第 9 回数値流体力学シンポジウム講演論文集，235(1995)．
[5]富士通 UXP/M VPP FORTRAN77EX/VPP 使用手引書 V12 用(1994)．
[6]Intel Paragon System Fortran System Calls Reference Manual(1995)．
[7] 折居茂夫：第 47 回全国大会講演論文集，情報処理学会，2D-4(1994)．
[8] 折居茂夫：情報処理学会研究報告，95-HPC-58，27(1995)．
[9] 富士通 UXP/M VPP アナライザ使用手引書 V10 用(1994)．
[10]根元俊行，他：JAERI-M 92-105，“新 FORTRAN コンパイラの導入とベクトル計算機の効果的利用方法”(1992)．
[11] 富士通 SSL II 使用手引書（科学用サブルーチンライブラリ）(1987)．
[12]Intel I860 Microprocessor Family Programmer's Reference Manual(1992)．
[13]Intel Paragon System Fortran Compiler User's Guide(1995)．
[13]大田敏郎：JAERI-Data/Code 96-009，“疎結合スカラ並列計算機のグローバル総和の高速化”(1996)．
[14]Lomdahl P. and Beazley D.:Los Alamos Science High-Performance Computing, No.22,44(1994)．

謝　辞

日本原子力研究所計算科学技術推進センターの渡辺正氏には，MD コードを使用させていただいたことを感謝いたします．同じく計算科学技術推進センターの蕪木英雄氏には，MD コードの物理的内容のレクチャーをいただいたことを感謝いたします．計算科学技術推進センターの広田章氏には，一部の時間測定をしていただいたことを感謝いたします．同じく計算科学技術推進センターの南正雪氏には，使用した MD コードのプログラム構造について議論をしていただいたことを感謝いたします．

参考文献

[1]折居茂夫：第 49 回全国大会講演論文集，情報処理学会，5T-01(1994)．
[2]Matsumoto, K. and Orii, S. ： 第 47 回全国大会講演論文集，情報処理学会，2D-5(1993)．
[3]神林奨：JAERI-M 92-080，“分子動力学シミュレーションコード ISIS の開発”(1992)．
[4]渡辺正，蕪木英雄，町田昌彦：第 9 回数値流体力学シンポジウム講演論文集，235(1995)．
[5]富士通 UXP/M VPP FORTRAN77EX/VPP 使用手引書 V12 用(1994)．
[6]Intel Paragon System Fortran System Calls Reference Manual(1995)．
[7] 折居茂夫：第 47 回全国大会講演論文集，情報処理学会，2D-4(1994)．
[8] 折居茂夫：情報処理学会研究報告，95-HPC-58，27(1995)．
[9] 富士通 UXP/M VPP アナライザ使用手引書 V10 用(1994)．
[10]根元俊行，他：JAERI-M 92-105，“新 FORTRAN コンパイラの導入とベクトル計算機の効果的利用方法”(1992)．
[11] 富士通 SSL Ⅱ使用手引書（科学用サブルーチンライブラリ）(1987)．
[12]Intel I860 Microprocessor Family Programmer's Reference Manual(1992)．
[13]Intel Paragon System Fortran Compiler User's Guide(1995)．
[13]大田敏郎：JAERI-Data/Code 96-009，“疎結合スカラ並列計算機のグローバル総和の高速化”(1996)．
[14]Lomdahl P. and Beazley D.:Los Alamos Science High-Performance Computing, No.22,44(1994).