プラズマ粒子コードのためのベクトル並列計算法

折 居 茂 夫*

(日本原子力研究所計算科学技術推進センター)

A Vector-Parallel Mapping Algorithm for Plasma Particle Codes

ORII Shigeo*

*Center for Promotion of Computational Science & Engineering,
Japan Atomic Energy Research Institute, Tokyo 153-0061, Japan*

(Received 20 August 1998/Accepted 5 April 1999)

Abstract

A parallel mapping algorithm for plasma particle codes on vector-parallel computers is shown with the use of an extended Abe-Nishihara vectorization algorithm. It is scalable not only for performance but also for size of problem. A plasma particle code employing this algorithm achieves scalable parallel performance on the distributed-memory-type vector-parallel computer VPP300 and the SX-4, the shared- and distributed-memory-type vector-parallel architecture. On such a code, parallel performance is determined by four ratios. First is a ratio between the number of arithmetic operations and the number of elements which are to be transferred to other processors. Second is a peak ratio between the arithmetic flop/s and communication MB/s performance. Third is a ratio between efficiency of the arithmetic and communication performance. Fourth is a ratio of the number of meshes to particles. Among those four ratios, the three ratios except for the third one are determined uniquely by the property of the code or computer. Therefore, the third ratio is an important issue to design and select parallel computers for the plasma particle codes.

Keywords:

gyro3d, gyrokinetic particle simulation, particle simulation code, PIC, particle in cell, vector computer, parallel computer, vector parallel computer, distributed memory, MPI

1. はじめに

プラズマの物理現象の解明に、PIC法 (Particle In Cell method) によるプラズマ粒子シミュレーションを用いることは古くから行われている。現在では、PIC法をプラズマ核融合炉の数値実験に用いることが日米で試みられている[1,2]。このような大規模な数値実験には、現存するスーパーコンピュータの100倍以上の処理能力とメモリを必要とするため、計算と粒子データを各

プロセッサに分割して処理する並列計算機の利用が不可欠である。並列計算機には各プロセッサがメモリを共有する「共有メモリ型」と各プロセッサ間を通信で結合する「分散メモリ型」が存在するが、核融合プラズマ数値実験で想定されるような大規模な粒子データの場合は、プロセッサ数をより多く接続できる分散メモリ型アーキテクチャを取り込んだ計算機を選択せざるを得ない。そして明らかに、通信を伴うデータの授受は、メモリを介

* Present Address: FUJITSU Limited, Chiba 261-8588, Japan
author's e-mail: orii@strad.se.fujitsu.co.jp

してのそれと比較して遅くなる。このような状況下で、大規模な数値実験が可能な並列計算機とはどのようなものか、PIC法を用いたプラズマ粒子シミュレーションコード（以後、プラズマ粒子コードと呼ぶ）とはどのような並列計算法を持つべきかを議論し、その実現に向けて取り組むことが重要な課題となっている。

プラズマ粒子コードの並列計算法を大別すると、物理空間を各プロセッサに分割して負荷分散を図る「領域分割法」と、粒子番号を各プロセッサに分割して負荷分散を図る「粒子分割法」に分類することができる。

領域分割法を用いると、電荷の寄与する遮へい距離を用いて通信範囲を限定できるため、粒子分割法に比べ通信量を少なくできる。一方、領域分割法は、プラズマの形状や領域の再分割等により各領域の粒子数が均等でない場合が生じるため、領域を再分割し各プロセッサ間の計算処理を均等にすることが必要となる。領域分割法概念図を Fig. 1 (a) に示す。図は二次元の x - y 空間にあるプラズマ粒子が近傍の領域ごとに4台の PE (Processing Element) に分割された様子を示す。PE間の線は隣接したPE間通信のためのネットワークである。領域分割法では、遮へい距離を考慮すると、隣のPEと通信するだけで電荷密度計算を行うことが可能となる。

粒子分割法を用いると、負荷分散を図ることが容易となり効率の良い負荷分散が可能となる。一方、時間ステップを進めるごとに電荷密度計算において全プロセッサ間にまたがる大規模な通信が発生するため、高速な通信性能を必要とする。粒子分割法概念図を Fig. 1 (b) に示す。粒子分割法では、プラズマ粒子はその座標に関係な

く4台のPEに同じ粒子数で分割される。図中では矢印でそのマッピングのようすを示す。図では煩雑さをさけるため、PE1とPE4についてのみ矢印を付けた。このように位置と無関係に粒子をマッピングすると、あるPEは他のすべてのPEとの通信が必要となり、高速でかつプロセッサ間の通信距離が均等なネットワークが必要となる。図中のクロスバーネットワーク (XBAR) はその要求を満たす一例である。

並列計算機でこれらの計算法を実現する場合、現在我々はPEとしてベクトルプロセッサあるいは、スカラプロセッサを選択することができる。このうち、分散メモリ型のスカラプロセッサを用いた並列計算機に対して領域分割法を適用する試みは、数多く行われてきた。この原因は、1980年代の早い時期から数多くの分散メモリ型のスカラ並列計算機が製品化されたこと。当時100から1,000台規模のスカラプロセッサを、たとえばメッシュのような通信時間がプロセッサ間の距離により変化する比較的簡単なネットワーク形状を用いて接続するという、MPP (Massively Parallel Processing) が流行したためと考えられる。比較的簡単なネットワーク形状を用いて接続する場合、近くに接続されているプロセッサとの通信が高速である。プラズマ粒子コードで扱うような遮へい距離を持つ計算で領域分割法を用いると、隣のプロセッサへの通信が支配的になるため、このような並列計算機と領域分割法がよくマッチすることが確かめられており [3-5]、最近ではジャイロカイネティック理論を用いたプラズマ粒子コード gyro3d で用いられている [6]。

一方1990年代半ばになって、分散メモリ型のベクトル

Fig. 1 (a) Domain decomposition algorithm. (b) Particle decomposition algorithm.

並列計算機に対する領域分割法の適用[5]と、粒子分割法の適用[7]が試みられた。これらに用いられた分散メモリ型のベクトル並列計算機が比較的最近実用化されたため (VPP500:1992年, SX-4:1994年出荷開始) と考えられる。これらの特徴は、クロスパーネットワークを採用してプロセッサ間の通信に距離の違いがない点、通信性能がそれまでのスカラ並列計算機に比べて高くなっている点であり、これは先ほど述べた粒子分割法に必要な通信の条件を満たしており、スケーラブルな性能向上の報告がある[7]。しかしながら、ベクトル計算のために用いられているワークベクトル法と呼ばれるベクトル計算法[8,9]は、ワークベクトルとして使用する巨大な配列を各プロセッサに分割できないため、本来のシミュレーションの規模を圧迫する欠陥がある。また一部に並列計算の効果がでない部分があり、これが時として性能に悪影響を与える。そこでこれらの制約を受けない粒子分

割法のベクトル並列計算法を研究し、その適用可能性を検討することにした。

本論文では、2章でデータマッピングの見地からプラズマ粒子コードの計算処理をまとめ、ベクトル計算法を示す。3章でベクトル並列計算法を議論し、ベクトル並列計算における従来のベクトル計算法の問題点を指摘する。4章で従来のベクトル計算法の問題点を解決するベクトル並列計算法を示す。5章で3章と4章で述べた粒子分割法を用いてベクトル並列化した gyro3d コードを用いて、ベクトル並列計算機 VPP300, SX-4の性能を評価する。

2. プラズマ粒子コード

典型的なプラズマ粒子コードのフローチャートを Fig. 2 に示す。コードの計算内容は、粒子電荷のメッシュへの分配、場の計算、粒子加速、および t_{obs} までの

Fig. 2 Flowchart and data mapping pattern of a typical plasma particle simulation code.

観測から成る。

これらの処理はデータマッピングの観点から、次の [I] から [IV] の 4 つの計算に分類することができる。[I] は電荷等の粒子データ B を粒子の位置 IX を基にしてメッシュ上 F に集める粒子分配と観測、[II] は場 F の計算、[III] は場の量 F を粒子の位置 IX に内挿する粒子加速、[IV] は粒子加速および観測である。

[I] 粒子からメッシュへのデータマッピングを含む計算

$$F(IX(N)) = F(IX(N)) + B(N) \quad (N=1, N_{\max})$$

[II] メッシュ上における計算

$$F(I) = FD(I) + A(I) \quad (I=1, I_{\max})$$

[III] メッシュから粒子へのデータマッピングを含む計算

$$P(N) = F(IX(N)) + B(N) \quad (N=1, N_{\max})$$

[IV] 粒子上における計算

$$P(N) = P(N) + B(N) \quad (N=1, N_{\max})$$

ここに、 F , FD と A はメッシュデータ、 P と B は粒子データ、 IX は粒子の位置、 $I_{\max}$ はメッシュ数、 $N_{\max}$ は粒子数を表す。核融合プラズマ粒子シミュレーションでは、セル中の粒子数 N_{cell} ($:= N_{\max}/I_{\max}$) が 10~100 を想定できる。以上のことを用いて、プラズマ粒子コードのベクトルおよびベクトル並列計算法を議論することができる。

2.1 ベクトル計算法

[I] から [IV] に分類された計算をベクトル計算法という観点から考察する。[II] は、配列要素 FD と A の和算であり、ベクトル計算可能である。これを Fig. 3 に示す。同様に [IV] もベクトル計算可能である。[III] は配列要素 IX から、配列要素 F を内挿して、配列要素 P を結果とする。Fig. 4 に示すように、[III] の右辺の第一項は場のデータをベクトル要素とすることができ、ベクトル計算可能である。これらのマッピングを含むベクトル計算は、現存する商用ベクトル計算機上で自動ベクトルコンパイラを用いて実現可能である。

一方、[I] は、 IX から F を内挿して同じ F にデータマッピングをする。このマッピングでは、マッピングしたデータを次の計算で使用する場合があります、その場合はベクトル計算ができない。この典型的な状態、 $IX(1) = IX(2) = \dots IX(I_{\max}) = 1$ を Fig. 5 に示す。図中矢印で示したように $F(1)$ の計算は左辺の計算結果を右辺の下の方に反映させる必要があり、ベクトル計算ができない。[I] の計算を Fig. 5 に示されたようにそのままベクトルで計算すると、計算結果が [I] で意図したことと一致

$$\begin{pmatrix} F(1) \\ F(2) \\ \vdots \\ F(I_{\max}) \end{pmatrix} = \begin{pmatrix} FD(1) \\ FD(2) \\ \vdots \\ FD(I_{\max}) \end{pmatrix} + \begin{pmatrix} A(1) \\ A(2) \\ \vdots \\ A(I_{\max}) \end{pmatrix}$$

Fig. 3 Typical field calculation on mesh using vector processing.

$$\begin{pmatrix} P(1) \\ P(2) \\ \vdots \\ P(N_{\max}) \end{pmatrix} = \begin{pmatrix} F(IX(1)) \\ F(IX(2)) \\ \vdots \\ F(IX(N_{\max})) \end{pmatrix} + \begin{pmatrix} B(1) \\ B(2) \\ \vdots \\ B(N_{\max}) \end{pmatrix}$$

Fig. 4 Typical calculation of particles including mapping field data onto particle data.

$$\begin{pmatrix} F(1) \\ F(1) \\ \vdots \\ F(1) \end{pmatrix} = \begin{pmatrix} F(1) \\ F(1) \\ \vdots \\ F(1) \end{pmatrix} + \begin{pmatrix} B(1) \\ B(2) \\ \vdots \\ B(N_{\max}) \end{pmatrix}$$

Fig. 5 Charge deposit calculation [I] including mapping particle data onto mesh, which can not use vector processing.

しない。このような計算を回帰計算と呼ぶ。この回帰計算は、 IX がすべて異なる値を持つ時は生じないため、ベクトル計算することができる。しかしプラズマ粒子コードは $N_{\text{cell}} > 1$ という特徴を持つため、 F の一要素が P の複数に対応するデータマッピングが必ず存在し、[I] は回帰計算となる。このため [I] のデータマッピングをベクトル計算で行うアルゴリズムの研究が行われてきた [8-12]。

これらの方法の中で、通常利用される作業領域を使用するもの [8,9] の概念を Fig. 6 に示す。ベクトル計算するため、計算結果を作業配列 $wF(LR, I_{\max})$ にマッピングし、Step2 でメッシュデータ $F(I_{\max})$ に再マッピングする。この方法は「ワークベクトル法」と呼ばれて

Step1

do $n = 1, N_{\max}, LR$

$$\begin{pmatrix} wF(IX(n),1) \\ wF(IX(n+1),2) \\ \vdots \\ wF(IX(n+LR-1),LR) \end{pmatrix} = \begin{pmatrix} wF(IX(n),1) \\ wF(IX(n+1),2) \\ \vdots \\ wF(IX(n+LR-1),LR) \end{pmatrix} + \begin{pmatrix} B(n) \\ B(n+1) \\ \vdots \\ B(n+LR-1) \end{pmatrix}$$

end do

Step2

do $L = 1, LR$

$$\begin{pmatrix} F(1) \\ F(2) \\ \vdots \\ F(I_{\max}) \end{pmatrix} = \begin{pmatrix} F(1) \\ F(2) \\ \vdots \\ F(I_{\max}) \end{pmatrix} + \begin{pmatrix} wF(1,L) \\ wF(2,L) \\ \vdots \\ wF(I_{\max},L) \end{pmatrix}$$

end do

Fig. 6 Non-recursive charge deposit calculation using the work-vector method.

いる。容易にわかるように、Step2 の計算は [I] にはない余分な計算であるが、この計算はベクトル計算機上で非常に高速に処理されるため、Fig. 6 の全体のベクトル計算は、[I] の計算を回帰的に逐次計算する場合より短時間で終了する。

ここに、パラメータ LR はベクトル計算計算機毎に最適値が異なり、通常数十から数百の値が用いられる。このためワークベクトル法用の配列 $wF(LR, I_{\max})$ が大規模になり、メモリを圧迫するという問題点が指摘されているが、その優れた性能ゆえ、単一プロセッサのベクトル計算機ではよく用いられてきた。

3. 粒子分割法を用いた並列計算法

並列計算法に粒子分割法を用いたプラズマ粒子コードは、1章で述べたように、粒子数を各プロセッサに均等に分割するため、各プロセッサの計算時間がほぼ同じになり、優れた負荷分散を実現できる。プロセッサ数を p とし粒子数を $N_{\max}$ とすれば、2章で述べた [I], [III], [IV] の計算を $1 \sim N_{\max}/p, N_{\max}/p + 1 \sim 2 \times N_{\max}/p, \dots$ と

各プロセッサに分割して並列計算できる。この粒子分割法は、例えば Fortran のように記述するのであれば $\text{do } 1, N_{\max}/p, \text{ do } N_{\max}/p + 1, 2 \times N_{\max}/p, \dots$ と、原則的にループの計算範囲のみを変更して各プロセッサに負荷分散すれば実現できるので、プログラミング的にも非常に容易である。

Fig. 7 に本論文で述べる粒子分割法による並列計算の処理の流れを示す。並列計算は網がけした部分、2章で述べた粒子計算部 [I], [III], [IV] で行う。通信は、粒子分配の直後のプロセッサ間にまたがる総和計算と、観測のエネルギーの総和計算にのみに必要となる。

一般に並列性能を阻害する主な要因は、通信時間、並列化率 (= (並列計算可能な部分の実行時間 / 全実行時間) $_{p=1}$)、負荷分散である。このうち負荷分散は、粒子分割法を採用したため主な要因から除外できる。粒子計算 [III], [IV] の並列化率は 100 % である。また 4 章に示す阿部-西原の考案したベクトル計算法 [11, 12] (以後、阿部-西原の方法と呼ぶ) を [I] に適用した場合、粒子計算 [I] の並列化率も 100 % である。したがって、粒子

分割法に阿部－西原のベクトル計算法を適用したプラズマ粒子コードの並列性能を決定する最大の要因は、通信となる。そこで通信の全粒子計算時間に占める影響を式(1)のように定義して議論する。

$$\frac{(\text{粒子計算時間})_{p=1}}{((\text{粒子計算時間})_{p=1} + (\text{通信時間})_p)} = 1/(1 + \gamma_p) \quad (1)$$

ここに、 $\gamma_p = (\text{通信時間})_p / (\text{粒子計算時間})_{p=1}$ である。粒子計算時間は、 $p=1$ の値を基準として用いることにする。また通信時間は、プロセッサ数に依存し、通信方式や通信アルゴリズム等により変化するため、 p の関数として記述した。

ここで $(\text{通信時間})_p$ を $\alpha(p) \cdot I_{\max} / (e_c \cdot r_c / 8)$ 、 $(\text{粒子計算時間})_{p=1}$ を $\beta(1) \cdot N_{\max} / (a_u \cdot r_a)$ とすると、 γ_p は式(2)のように4つの比の積で表すことができる。

$$\gamma_p = (\alpha(p) / \beta(1)) \cdot (a_u / e_c) \cdot (r_a / r_c / 8) \cdot (N_{\text{cell}})^{-1} \quad (2)$$

ここに、 $\alpha(p)$ はFig.7の[I]直後の通信量で、電子とイオンの電荷と電流等の場の種類と通信方式で決まる。 r_c (MB/s)は通信の最大性能、 e_c はその効率である。 $r_c/8$ は1秒間あたりに通信できる要素数である。 $\beta(1)$ は $p=1$ のFig.7の[I], [III], [IV]のループ中の四則演算数である。 r_a (Gflop/s)は四則演算の最大性能で、 a_u はその効率である。以前に示したように $N_{\text{cell}} = N_{\max} / I_{\max}$ である。 γ_p は N_{cell} に反比例することから、 N_{cell} が大きい計算ほど通信時間の割合が減少することがわかる。プラズマ核融合の数値実験では通常 $N_{\text{cell}} > 10$ である。

高い並列効果を得るためには式(1)の値を1に近づけることが必要で、そのためには γ_p の値が低くなることが要求される。しかし式(2)は、四則演算の効率 a_u が低い場合にも γ_p の値が低くなることを示す。この場合、計算がプロセッサの最大flop/s性能に見合わない低い性能で実行されると相対的に通信時間の割合が減少する結果、スケラビリティを見るとリニアとなり、一見並列性能がよく見える。この現象を識別することは、ただプロセッサ数を変えて計算時間を測定しただけではできず、並列性能の原因である a_u と e_c を導入することにより初めて識別することができる。 a_u と e_c は、使用した計算法がベクトル計算できない場合や通信競合を起こす場合などの本質的な問題のため低い場合もあるが、少しのコーディングの変更でコンパイラの最適化方法が変

Fig. 7 Flowchart of a plasma particle code using particle decomposition parallel algorithm.

わり、大きく変わる場合もある。

ところでFig.7の観測[IV]では、 $N_{\max}$ に比例した通信量が発生する。しかし全体の時間積分回数に対する t_{obs} ごとの観測は頻度として低く、通常無視することができる。また、場の計算[II]は解法により並列計算法が異なる。本論文で述べるgyro3dはFFTを用いている。FFTは並列化計算できることが知られている[13]。またBi-CG法が用いられている場合もあり[7]、粒子分割法とは独立に議論できるので、本論文では触れない。

4. 阿部－西原のベクトル計算法を用いた粒子分割法

2章の[I]にワークベクトル法を用いると、Fig.6のStep2に並列計算の効果がでない。Step2の計算回数 $I_{\max} \times LR$ は、プロセッサ数が増加しても一定だから

である。このため、 LR に数百の大きな値が適する場合、並列計算によるスケーラブルな性能向上が望めない。 LR の最適値は、計算の立ち上がり時間（ベクトル計算速度が定常になるまでの時間）、 $N_{\text{cell}}, I_{\text{max}}$ により決まる[14]。計算の立ち上がり時間は、一般にプログラミング方法や計算機により異なる。実際 gyro3d にワークベクトル法を適用すると LR の最適値は512となり、Step2 が Fig. 6 の計算時間の数十%を占め、スケーラビリティは数台のプロセッサで失われる。また各プロセッサで $I_{\text{max}} \times LR$ の配列が必要となり、巨大なワーク配列を必要とする。この2点が、ワークベクトル法を用いた場合の問題点である。これらの問題点は、3章で述べた、阿部-西原の方法を改良して並列化することにより解決できる。

阿部-西原の方法は、2章の[1]を IX の値が異なる要素のみでベクトル計算する方法である。原理を Fig. 8 に示す。例として、図では $IX(n) = IX(n+1) = IX(n+2)$ で、そのほかの IX はすべて異なった値を持つとした。Step1 のベクトル計算をベクトル計算機で行うと、 $F(IX(n))$ が $F(IX(n+1))$ で、 $F(IX(n+1))$ が $F(IX(n+2))$ で重ね書きされ、 $F(IX(n+2))$ の値のみが反映される。そこで、 $IX(n)$ と $IX(n+1)$ のテーブルを

作成し、Step2 では、Step1 で計算できなかった $IX(n)$ と $IX(n+1)$ のデータを対象にベクトル計算を行い、 $F(IX(n+1))$ の計算結果を加える。

Step3 で $F(IX(n))$ の計算結果を加えて計算が完了する。この例では、一回のベクトル計算で同じ IX が3個なので、3回の繰り返し計算する。

当然ながらStep2, Step3はオーバーヘッドとなる。プラズマ粒子コードは $N_{\text{cell}} > 1$ であり、粒子番号と位置に相関関係があると、この繰り返しが生じる確率が増える。核融合プラズマでは、 N_{cell} が数十から百の大きさとなり、かなり高い確率となる。このため阿部-西原の方法をそのまま使用すると、ベクトル計算による性能向上が繰り返し計算のオーバーヘッドのため低減し、性能が向上しない場合がある。

この問題点は、粒子分配が粒子の近傍に位置するメッシュにのみに限られる性質を利用して解決できる。すなわち、粒子の位置 $IX(n)$ と粒子番号 n との相関をなくすように IX を乱数により並べ替える。これにより隣り合った粒子番号の粒子が距離的に離れることができる。一回にベクトル計算する粒子数を $LR \sim$ 数百、 $N_{\text{max}} > 10^5$ とすると、 LR/N_{max} は限りなく零に近づき、Step2 以降の繰り返し計算はほとんど生じない。このよ

Step1

do $n=1, N_{\text{max}}, LR$

$$\begin{pmatrix} F(IX(n)) \\ F(IX(n+1)) \\ F(IX(n+2)) \\ \vdots \\ F(IX(n+LR-1)) \end{pmatrix} = \begin{pmatrix} F(IX(n)) \\ F(IX(n+1)) \\ F(IX(n+2)) \\ \vdots \\ F(IX(n+LR-1)) \end{pmatrix} + \begin{pmatrix} B(n) \\ B(n+1) \\ B(n+2) \\ \vdots \\ B(n+LR-1) \end{pmatrix}$$

end do

Step2

$$\begin{pmatrix} F(IX(n)) \\ F(IX(n+1)) \end{pmatrix} = \begin{pmatrix} F(IX(n)) \\ F(IX(n+1)) \end{pmatrix} + \begin{pmatrix} B(n) \\ B(n+1) \end{pmatrix}$$

Step3

$$(F(IX(n))) = (F(IX(n))) + (B(n))$$

Fig. 8 Conceptual view of Abe-Nishihara's vectorizable data mapping algorithm for charge deposition.

Step1

do $n=1, N_{\max}, LR$

$$\begin{pmatrix} F(IX(n)) \\ F(IX(n+3)) \\ F(IX(n+4)) \\ \vdots \\ F(IX(n+LR-1)) \end{pmatrix} = \begin{pmatrix} F(IX(n)) \\ F(IX(n+3)) \\ F(IX(n+4)) \\ \vdots \\ F(IX(n+LR-1)) \end{pmatrix} + \begin{pmatrix} B(n) \\ B(n+3) \\ B(n+4) \\ \vdots \\ B(n+LR-1) \end{pmatrix}$$

end do

Step2

$$(F(IX(n+1))) = (F(IX(n+1))) + (B(n+1))$$

Step3

$$(F(IX(n+2))) = (F(IX(n+2))) + (B(n+2))$$

Fig. 9 The extended Abe-Nishihara vectorization algorithm.

うな計算はネストした二重ループで実現できる。具体的には Fig.10を参照されたい。

一方、上記性質を持たない計算に阿部-西原の方法を適用すると、 LR 個の粒子中で同じメッシュ点への粒子分配が避けられなくなることがある。その場合 Step2以降の繰り返し計算が増加し、この計算法の効果が期待できない。例えば強結合プラズマではその可能性が高い。

Fig. 8 の阿部-西原の方法はベクトル計算による重ね書きを想定している。このベクトル計算は Fortran の文法から逸脱した処理で、文法上は Fig. 5 の矢印のようにデータマッピングが行われるためこの重ね書きは生じない。このためベクトル計算機以外で Fig. 8 の阿部-西原の方法を適用すると、誤った計算結果となる。例えばスカラ計算機で阿部-西原の方法を実行する場合がこれにあたる。またこの重ね書きをベクトル計算で実現する場合、1回のベクトル命令の実行で LR 個の要素数のベクトル計算をするように、オブジェクトコードを意識したコーディングが必要になる。一般にベクトル化された do ループの計算は、オブジェクトコードにおいてベクトル命令を複数回繰り返すように記述される。Fig. 8 の Step1 は、オブジェクトコードにおいては $1 \sim LR, LR+1 \sim 2 \cdot LR, \dots$ のように LR 要素ごとのベクトル計算を繰り返すオブジェクトコードを用いる。しかし、Fortran 言語においてこれを保証する文法は存在しない。

い。

本論文ではこの問題を、重ね書きする代わりに、Fig. 9 のようにベクトル計算できる粒子要素を選別して計算することにより解決できることを示す。Fig. 9 の Step1 では、要素が n から $n+3$ に飛んで計算されていることに、注目されたい。

このように選別という機能を持つ、拡張された阿部-西原の方法を用いて粒子分割法で並列化したコーディングを Fig.10に示す。使用言語は Fortran、通信のための並列ライブラリには、MPI[15]を用いた。初めの4行の実行文と、GSUMの部分以外は、1プロセッサの計算と同じである。初めにこの共通部分を説明する。 $nLOCAL$ 個の粒子数を、 LR 要素数ごとにベクトル計算する。 LR の最適値として512を用いた。Step1の do $n = nSTART, nEND, LR$ の内側にある2つのループにおいて、初めの do ループでは、メッシュデータ $iT(I_{\max})$ に粒子の位置 IX の粒子番号 n を記録する。複数の粒子の IX が同じ番号の場合、 iT には最後に出現した粒子番号が選別される。二番目の do ループでこのテーブル iT を if 文で参照し、ベクトル計算できる選別された粒子番号を検出して $F(IX(nn)) = F(IX(nn)) + B(nn)$ を計算する。*vocl loop, novrec, と *vdir nodep は、各々 VPP300 と SX-4 に対応する、直後のループが回帰計算でないことを示すコンパイラ指示行で

ある。Iteration Step では、do while を使い、nct が 0 になるまで同じ処理を繰り返す。なお、単一プロセッサ NP = 1 では、nSTART = 1, nEND = $N_{\max}$ となり、GSUM は実行しないようにするのがよい。

並列計算では、はじめに各プロセッサが分担する粒子番号を決定する。最初の 4 行に実行文がこれにあたる。このためにあらかじめ MPI ライブラリ CALL MPI_COMM_SIZE (MPI_COMM_WORLD, NP, IERR), CALL MPI_COMM_RANK (MPI_COMM_WORLD, myRANK, IERR) をほかのサブルーチンにおいて順に

実行し、プロセス番号 myRANK を調べておく。本論文で myRANK は各プロセッサに対応し、0 から NP-1 までの整数で表される。myRANK と各プロセッサで扱う粒子数 nLOCAL を用いて、各プロセッサで分担する粒子番号の始点 nSTART, 終点 nEND を計算することができる。なお、MPI_COMM_WORLD と IERR の意味は、文献[15]あるいは各計算機のマニュアル等を参照されたい。図右側の 2 番目以後の Step では、各プロセッサは分担した粒子番号の一部を計算するため、プロセッサ数が増すと計算量が減少する。これが、

Step1	Iteration Step(s) and Communication
<pre> subroutine ANalg (Nmax, lmax, IX, & B, F, nT, NP, myRANK) c NP:The number of Processors c myRANK:Value of MPI_COMM_RANK real*8 B(Nmax), F(lmax) integer*4 IX(Nmax), iT(lmax), & nT(*) nLOCAL=(Nmax+NP-1)/NP nSTART=myRANK*nLOCAL nEND=min(nSTART+nLOCAL, Nmax) nSTART=nSTART+1 LR=512 nct=0 do n=nSTART, nEND, LR do nn=n, min(n+LR-1, nEND) iT(IX(nn))=nn end do *vocl loop, novrec *vdir nodep do nn=n, min(n+LR-1, nEND) if(iT(IX(nn)).eq.nn) then F(IX(nn))=F(IX(nn))+B(nn) else nct=nct+1 nT(nct)=nn end if end do end do </pre>	<pre> do while(nct.gt.0) nctd=0 do nnn=1, nct, LR do nn=nnn, min(nnn+LR-1, nct) n=nT(nn) iT(IX(n))=n end do c *vocl loop, novrec *vdir nodep do nn=nnn, min(nnn+LR-1, nct) n=nT(nn) if(iT(IX(n)).eq.n) then F(IX(n))=F(IX(n))+B(n) else nctd=nctd+1 nT(nctd)=n end if end do end do nct=nctd end do return end call GSUM(lmax, F, FW, NP) </pre>

Fig. 10 Parallel coding of the extended Abe-Nishihara vectorization algorithm by Fortran and MPI communication library.

Step2 でプロセッサ数が増しても常に $I_{\max} \times LR$ の計算を行い計算量が変わらない、ワークベクトル法との相違である。GSUM では、通信を伴うプロセッサ間の総和 $F = \sum_{p=1}^{NP} F_p$ を行う。このコーディングを Fig.11 に示す。ここで NP はプロセッサ数である。Fig.11 を実行後、各プロセッサの F は同じ値となる。この処理は、ワークベクトル法でも必要な処理である。MPI_ALLTOALL で記述したこの処理は MPI ライブラリの MPI_ALLREDUCE で記した処理と同じ機能を持つが、多くの計算機において現在、この記述方法の方が高速である。

拡張された阿部-西原の方法では、ベクトル、スカラー計算機を問わず正しい計算結果を得ることができる。GSUM 以外のすべてが並列処理であり、スケーラブルな性能を期待できる。またワークベクトル法で用いた作業配列 wF が不必要となる。一方新たに作業配列 iT ($I_{\max}$)、 $nT(N_{\max})$ を必要とするが、 nT は $(N_{\max} + NP - 1)/NP$ の大きさで各プロセッサに分割することができる。さらに $B(N_{\max})$ と $IX(N_{\max})$ を各プロセッサに分割すれば、問題の規模をプロセッサ台数にスケラブルに大きくすることが可能となる。

```

subroutine GSUM(Imax, F, FW, NP)
  real*8 F(Imax), FW(Imax)
  include 'mpif.h'

c  call MPI_ALLREDUCE(F, FW, Imax, MPI_DOUBLE_PRECISION, MPI_SUM,
c  &                    MPI_COMM_WORLD, IERR)

  nLOCAL=(Imax+NP-1)/NP

  call MPI_ALLTOALL(F, nLOCAL, MPI_DOUBLE_PRECISION,
&                    FW, nLOCAL, MPI_DOUBLE_PRECISION,
&                    MPI_COMM_WORLD, IERR)

  do j=2, NP
    k=(j-1)*nLOCAL
    do i=1, nLOCAL
      FW(i)=FW(i)+FW(i+k)
    end do
  end do

  do j=2, NP
    k=(j-1)*nLOCAL
    do i=1, nLOCAL
      FW(i+k)=FW(i)
    end do
  end do

  call MPI_ALLTOALL(FW, nLOCAL, MPI_DOUBLE_PRECISION,
&                    F, nLOCAL, MPI_DOUBLE_PRECISION,
&                    MPI_COMM_WORLD, IERR)
  return
end

```

Fig. 11 Global sum between all processors with Fortran and MPI liberally.

5. ベクトル並列計算機における粒子分割法の性能評価

拡張された阿部-西原の方法を用いた粒子分割法で gyro3d コードを並列化し、日本原子力研究所中目黒地区に設置されたベクトル並列計算機 VPP300 と SX-4 の並列性能を評価した。VPP300 は、ピーク性能2.2 Gflop/s のプロセッサを、プロセッサ間通信性能570 MB/s のクロスバスイッチで結合した、15プロセッサのシステムである。SX-4 は、ピーク性能2 Gflop/s のプロセッサを、ノード内を共有結合で、ノード間をクロスバスイッチで結合し、メモリ間アクセス性能8GB/s の6プロセッサ (= 3ノード・2 (プロセッサ/ノード)) システムである。問題の規模は、 $N_{\max} = 128 \times 128 \times 64$, $I_{\max} = 32 \times 32 \times 16$ を用いた。 $N_{\text{cell}} = 64$ である。

Fig.12に VPP300 と SX-4 で実測した gyro3d の並列性能を示す。図中 $\tau(p)$ はプロセッサ数 p のときの実測値を表す。並列効率 $Ep (= \tau(p=1)/\tau(p)/p)$ を定義すると、図はプロセッサ数が5台位の規模では $Ep = 80\%$ 以上で、10台では $Ep = \text{約}70\%$ の効率で並列性能が得られることを示す。測定範囲での VPP300 と SX-4 の並列効率はほぼ同じプロセッサ依存を示す。なお、VPP300 のコンパイラは Fortran90/VP (L97121), コンパイルは `f90 -P -L /usr/lang/mpi/lib -lmpi -lmp -lpx` で行った。SX-4 のコンパイラは f90 compiler Rev.093, コンパイルは `f90 -P multi -Wf -pvct1 loopcnt=1048576 -c -I /usr/include -lmpi` で行った。

Fig. 12 Parallel performance of gyro3d using the extended Abe-Nishihara vectorization algorithm adapted to particle decomposition algorithm in case of $N_{\text{cell}} = 62$.

Fig.13に VPP300 の粒子計算時間 (2章の[I], [III], [IV]の測定値の合計)に関するスケラビリティを示す。GSUM を含まない場合、 $\tau_{\text{particle}}(p=1)/\tau_{\text{particle}}(p)$ は、ほぼ理想値と同じである。GSUM を含む場合、 $(\tau_{\text{particle}}(p=1) + \text{GSUM})/(\tau_{\text{particle}}(p) + \text{GSUM})$ は、プロセッサの増加とともに幾分理想値からずれるものの、粒子分割法とベクトル並列計算機の組み合わせが有効であることが確認できる。

Fig.13のように粒子分割法とベクトル並列計算機の組み合わせが有効であるか否かは、式(2)の γ_p を評価することにより知ることができる。

まず四則演算の効率 a_u と通信の効率 e_c は、各々式(3), (4)から計算する。

$$a_u = \beta(1) \cdot I_s \cdot N_{\max} / r_a / \tau_{\text{particle}} \quad (p=1) \quad (3)$$

$$e_c = \alpha(p) \cdot I_{\max} \cdot I_s / r_c / (\text{GSUM}(p) - \text{GSUM}(1)) \quad (b>1) \quad (4)$$

ここに、 $\beta(1) = 1248$, I_s は時間ループの繰り返し回数, $\tau_{\text{particle}}(p=1)$ は1PEで実行したときの実測粒子計算時間である。 r_a は Mflop/s で表した最大性能である。また文献[16]より $\alpha(p) = 2 \cdot 2 \cdot (p-1)/p \cdot 5$ となる。初めの2は予測子・修正子の2回実行, 次の2は ALLTOALL の2回実行, $(p-1)/p$ は自分以外のプロセッサに送るデータの割合, 5は5変数の GSUM であることを意味する。GSUM(p) はプロセッサ数 p の時の Fig.11全体の

Fig. 13 Total parallel performance of calculations related to particle.

実測時間を, $\text{GSUM}(p)\text{-GSUM}(1)$ は, $I_{\max} \gg p$ を仮定して図中の総和計算時間を差し引いた, p の時の通信時間である. r_c は B/s であらわした最大性能である.

Table 1 に VPP300 と SX-4 に対する a_u と e_c を示す. なお e_c は, VPP300 では $p=14$, SX-4 では $p=6$ で計算した. a_u/e_c を計算式(2)に代入すると式(5), (6)を得る. 右辺の3つの係数は各々 (α/β) , (a_u/e_c) , $(r_a/r_c/8)$ に対応する.

$$\begin{aligned} [\gamma_p=14]_{\text{VPP300}} \\ &= 0.0149 \cdot 0.647 \cdot 30.9 / N_{\text{cell}} \\ &= 0.30 / N_{\text{cell}} \end{aligned} \quad (5)$$

$$\begin{aligned} [\gamma_p=6]_{\text{SX-4}} \\ &= 0.0134 \cdot 16.9 \cdot 2.00 / N_{\text{cell}} \\ &= 0.45 / N_{\text{cell}} \end{aligned} \quad (6)$$

式(5), (6)はプラズマ核融合炉の数値実験のように N_{cell} が大きい場合, VPP300 と SX-4 で通信の寄与を数パーセント以下に押さえることができることを示す. gyro3d では四則演算数が多いため (α/β) は 0.01 のオーダーとなる. 計算機の仕様の差で $(r_a/r_c/8)$ の値は2つの機種では大きく異なるが, 効率の比 (a_u/e_c) によりこれらが相殺し, 係数は 0.30 と 0.45 となった.

Fig.12 で示す gyro3d コード全体の並列性能が, 粒子計算部分の並列性能 (Fig.13) より劣る理由は, Fig.14 から知ることができる. Fig.14 中の Init. は初期計算, field は2章の(2)で示した場に関する計算, particle は2章の[I], [III], [IV]で示した粒子に関する計算, GSUM は[I]の粒子から場へのマッピングした直後に行う通信を含む総和計算である. 図は並列処理の結果 particle の処理時間が減少し, 並列化されていない field, GSUM, Init. の比率が高くなり, それが原因でスケラビリティが低下していることを示す. また図は, プロセッサ数の増加に伴い GSUM が増加すること, す

なわち γ_p が増加することを示す.

このうち, Init. は初期に一回計算されるだけであるから, タイムステップの増加とともに無視できる時間となる. したがってスケラビリティを理想に近づけるためには, field の計算時間の削減すなわち並列化が必要となる. 3章で述べたように, gyro3d では field に FFT を用いる. FFT は並列化が可能のため, この部分の処理時間は削減可能である.

6. 議論

四則演算の効率 a_u と通信の効率 e_c はプログラムのコーディングに依存する. 例えば, Fig.11 でコメントにした MPI_ALLREDUCE を使用した場合, MPI_ALLTOALL を使用したコーディングに比べ長い処理時間を必要とする. これは主に a_u と e_c が両方で異なるためである. a_u と e_c は, ハードウェアをいかに活用するかを決めるコンパイラと通信ライブラリの処理能力に依存して決まる. したがってコンパイラと通信ライブラリのバージョンレベルが変わるにしたがい, 変化することも予測される. このため, 並列計算機の γ_p とはどのようなものを議論する場合, 実測値の比を評価するだけでは不十分で, 実現された性能がハードウェアの仕様とおりであるか否か, 否なら何が性能を決める原因か, コンパイラがハードウェアの処理能力を十分発揮するように働いているかをコーディングの変更などにより, 確認する必要がある.

a_u と e_c から性能を決める原因をある程度推測できる. Table 1 で示すように効率 a_u は低い. ベクトル計

Table 1 Uni-processing and communication performance related to particle in case of $I_s = 1000$, $N_{\max} = 128 \cdot 128 \cdot 64$, $I_{\max} = 32 \cdot 32 \cdot 16$ and $N_{\text{cell}} = 64$.

	VPP300	SX-4
a_u	0.11($r_a=2.2$ Gflop/s)	0.27($r_a=2$ Gflop/s)
e_c	0.17($r_c=570$ MB/s, $p=14$)	0.016($r_c=8$ GB/s, $p=6$)
γ_p	0.0047($p=14$)	0.0071($p=6$)

Fig. 14 Processing time of four typical calculations of gyro3d on VPP300.

算機に対するこの原因は、do ループの回転数が少ない、複数あるベクトル演算器を有効に利用できない、データをメモリからアクセスする時間が長く演算に追いつかないことの3つをあげることができる。gyro3d コードでは、do ループの回転数は512でありベクトル計算には十分である。VPP300 と SX-4 では足し算と掛け算の2つの四則演算器が同時に稼働したときピーク性能となる。particle の do ループ中の四則演算数は約1,248のうち、和算のみで演算器が同時に稼働できないと判断できる演算が144ヶ所存在するが、 a_{ii} を0.3以下にする根拠としては不足である。2章で示した $F(IX(N))$ のようなデータアクセスはインダイレクトアクセスと呼ばれ、ベクトル計算機の性能を低下させる原因となる。インダイレクトアクセスは particle 計算中に144ヶ所存在し、性能低下の原因の一つと推測できる。インダイレクトアクセスによる性能低下は、ハードウェアに依存するところが大きく、性能向上のためにはコーディングスタイルを現状のままにしてハードウェアを改良するか、ハードウェアを現状のままにしてコーディングスタイルを変更するかという本質的問題を含んでいる。一方 e_c の低下の原因解明は、通信の立ち上がり時間を考慮したより詳細な評価方法[16,17]の適用が必要と考える。

上記で議論したことを考慮しながら性能評価を継続しその結果を蓄積していくため、ベンチマークコード gyro3d-BMT[18]を開発した。gyro3d-BMT は、本論文で述べた並列計算法で粒子計算の部分と並列化してあるのに加え、各プロセッサへの粒子の分割マッピングおよび、FFT を用いた場の計算が並列化されている。このコードにより、粒子計算部分ばかりでなくプラズマ粒子コードとして総合的に並列計算機を評価することが可能と考える。

7. まとめ

プラズマ粒子コードのためのベクトル並列計算法を示した。粒子分割法に拡張された阿部-西原の方法を適用すると、粒子計算部分について性能がスケラブルに向い、各プロセッサに粒子データをスケラブルに分割マッピングできることを示した。この計算法をプラズマ粒子コード gyro3d に適用し、粒子計算部分におけるVPP300 と SX-4 における性能と問題の規模に対するスケラビリティを示した。性能に対するスケラビリティは、4つの比により決まり、性能評価の結果、そのう

ちの一つの四則演算効率と通信効率の比の評価が重要であることを示した。

謝辞

山口大学の内藤裕志氏、日本原子力研究所の徳田伸二氏には、プラズマ粒子コード gyro3d を使用させていただいたことを感謝いたします。

参考文献

- [1] 徳田伸二：プラズマ核融合学会誌. 72, 916 (1996).
- [2] B.J. Choen, D.C. Barnes, J.M. Dawson, G.W. Hammett, W.W. Lee, G.D. Kerbel, J.-N. Leboeuf, P.C. Liewer, T. Tajima and R.E. Waltz, *Comput. Phys. Comm.* **87**, 1 (1995).
- [3] B.L. Buzbee, *High Speed Computation* (Springer-Verlag, Berlin Heidelberg, Ed. J. S. Kowalik, 1984), p. 417.
- [4] P.C. Liewer and V.K. Decyk, *J. Comput. Phys.* **85**, 302 (1989).
- [5] 西原功修, 新宮 哲, 川口正仁, 岸本泰明, 谷 啓二：プラズマ・核融合学会誌. 72, 926 (1996).
- [6] H. Naitou, H. Kitagawa and S. Tokuda, *J. Plasma Fusion Res.* **73**, 174 (1997).
- [7] 田中基彦：プラズマ・核融合学会誌. 72, 542 (1996).
- [8] A. Nishiguchi, S. Orii, and T. Yabe, *J. Comput. Phys.* **61**, 519 (1985).
- [9] A. Heron, and J.C. Adam, *J. Comput. Phys.* **85**, 284 (1989).
- [10] E.J. Horowitz, *J. Comput. Phys.* **68**, 56 (1987).
- [11] Y. Abe, *Supercomputing'88* (1988) p. 72.
- [12] K. Nishihara, H. Furukawa, M. Kawaguchi and Y. Abe, *Japanese Supercomputing* (Springer-Verlag, NY, Lecture Notes in Engineering, 36, Eds. R.H. Mendez, S.A. Orszag, 1988) p. 59.
- [13] 例えば、福田正大, 吉田正廣, 中村 孝, 村瀬丈夫, 小山隆司：情報処理学会第47回(平成5年後期)全国大会 6H-9, 6-155 (1993).
- [14] S. Orii, *International Meeting on Advances in Nuclear Engineering Computational Methods*, Knoxville Tennessee, 22, April 9-11 (1985).
- [15] 例えば, MPI: A Message-Passing Interface Standard, Message Passing Interface Forum, June 12 (1995).
- [16] 折居茂夫：情報処理学会論文誌 **39**, 529 (1998).
- [17] 折居茂夫：JAERI-Data/Code 98-020 (1998).
- [18] 折居茂夫, 渡部 弘, 滝川好夫, 熊倉利昌, 久米悦夫：計算工学講演会論文集 **3**, 43 (1998).