収束性を考慮した動的負荷分散手法による
オイラー方程式の並列計算*小幡 正規^{*1}, 鈴木 智博^{*2}, 里 深信 行^{*1}Parallel Computations for Euler Equations
Using a Dynamic Load Balancing Technique

Masanori OBATA, Toshihiro SUZUKI and Nobuyuki SATOFUKA

A dynamic load balancing technique is applied for solving the Euler equations on networked engineering workstations. Message passing is handled by PVM (Parallel Virtual Machine) software. A domain decomposition method is employed to parallelize the algorithm. In this load balancing technique, the convergence criterion is evaluated on each subdomain. When the criterion is satisfied on a subdomain, the computational domain other than that subdomain is redecomposed on each processor. This procedure is performed once for supersonic flow, and several times for transonic flow. In the present approach, the total load of the calculation of two-dimensional Euler equations solvers is reduced by 48% for the test case of supersonic flow in a duct and by 29% for transonic flow.

Key Words: Computational Fluid Dynamics, Numerical Analysis, Finite Difference Method, Parallel Computation, Parallel Virtual Machine, Domain Decomposition Technique

1. 緒 言

計算流体力学の発達はめざましく、航空宇宙分野や自動車工学の分野をはじめとする多くの理工学分野においてその設計ツールとして実用化されつつある。しかしながらより現象に忠実に流れを計算するためには現在のスーパーコンピュータの性能では不十分であり、さらに高い性能のコンピュータが要望されているが、そのような高性能のコンピュータは現在のプロセッサアーキテクチャに何らかのブレイクスルーがないかぎり必然的に並列計算機となることが予態される。また、実際近年開発されたスーパーコンピュータはほぼすべてが並列計算機となっている。そのゆえ、並列計算機に適した流れの数値計算アルゴリズムの開発が急務となっている。現在までには、既存のプログラムを比較的容易に並列化することができる領域分割法を用いた研究が多く報告されている^{(1)~(3)}。しかし、それらは従来から用いられてきた手法を採用しているものおよび既存する手法を並列計算可能となるように若干の改良を

加えたものがほとんどであり、並列計算を行うことを前提に考案された計算アルゴリズムはまだ数少ない。

そこで、本論文では領域分割法を用いた並列計算に適合するように、定常問題に対して各領域での収束性の違いを考慮した動的負荷分散法に基づく収束加速法を提案し、本手法を二次元の遷音速流れおよび超音速流れを適用することにより計算量の軽減および並列計算効率をそれぞれの場合について調べることを目的とする。並列計算機向けに考案した本収束加速法は、領域分割法を用いれば単一のプロセッサにおいても利用可能である。本収束加速法では、収束の早い小領域を受けもつノードプロセッサでは計算を一時的に停止し、収束の遅いノードプロセッサの小領域を新たに受けもつことにより全体としての計算量の負荷を軽減し、計算時間を短縮させることを試みている。並列計算にはPVM(Parallel Virtual Machine)⁽⁴⁾を用い、仮想的に最大4台のワークステーションを並列計算機として使用している。

2. 並列計算機(PVM)の概要

2.1 PVM PVM⁽⁴⁾はネットワーク上のさまざまな計算機(ワークステーション)をあたかも一つの仮想計算機のプロセッサのように見立て並列計算をする

* 原稿受付 1995年6月20日。

^{*1} 正員, 京都工芸繊維大学工学部 (〒606 京都市左京区松ヶ崎御所海道町)。^{*2} ダイキン工業(株) (〒591 堺市金岡町1304)。

ことができる管理ソフトである。つまり、ネットワーク上のワークステーションを分散メモリの MIMD 形 (Multiple Instruction streams, Multiple Data streams) の並列計算機として使用できることになる。分散メモリの MIMD 形並列計算機ではそれぞれのプロセッサ (ここではワークステーション) に共有するメモリがなく、異なるプロセッサ間のデータの参照にはユーザが意識的に通信を行う必要が生じる。

本研究では、ホスト用のワークステーション 1 台に対して、並列計算用のノードとしてワークステーションを 4 台用いている。それぞれのワークステーションはイーサネット で接続されている。

2.2 イーサネットの通信時間 二つのプロセッサ間の単純な通信モデルでの実際の通信時間は次のようになる⁽⁵⁾。

$$T_{\text{comm}} = T_l + WT_w + T_t \quad \dots\dots\dots (1)$$

ただし、

T_{comm} : データ通信に必要な時間

T_l : レイテンシ (通信のための準備時間)

W : データ量

T_w : 単位データ当たりの転送時間

T_t : トポロジーによるオーバーヘッド

である。上式をふまえて実際にイーサネット上のプロセッサ間における通信時間を測定してみると図 1 のようになった。ただし図 1 の 1 PE, 2 PEs, 4 PEs とはそれぞれ 1 対 1, 1 対 2, 1 対 4 のプロセッサ間における通信時間を意味している。図 1 において通信量が 1 KByte 以下では通信に要する時間はほぼ一定であり、イーサネット で小さなデータを送信することは並列システムの効率の低下になることがわかる。その理由は式 (1) におけるレイテンシの時間が通信時間の大部分を占めているからである。また、プロセッサ数が多く

なるにつれて通信時間が大きくなっている。それはイーサネット 上でメッセージの衝突が生じればある時間間隔をランダムに決定し、その時間の後に再送出を行うことになり、式 (1) におけるトポロジーによるオーバーヘッドが大きくなるからである。通信速度を測定してみると、通常、イーサネット の通信速度は 10 Mbit/s といわれているが、プロトコルによるオーバーヘッドのために 1 KBytes 以上でも 2~3 Mbits/s 程度であった。

3. 基礎方程式および数値計算法

3.1 基礎方程式 基礎方程式として二次元非粘性圧縮性流を支配するオイラー方程式を用いる。オイラー方程式は Cartesian 座標 (x, y) から一般座標 (ξ, η) 座標変換を行うことにより次のように記述される。

$$\frac{\partial q}{\partial t} + \frac{\partial E}{\partial \xi} + \frac{\partial F}{\partial \eta} = 0 \quad \dots\dots\dots (2)$$

式 (2) において保存量ベクトル q , 流束ベクトル E および F は、以下のように表される。

$$q = \frac{1}{J} \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ e \end{bmatrix}, \quad E = \frac{1}{J} \begin{bmatrix} \rho U \\ \rho u U + \xi_x p \\ \rho v U + \xi_y p \\ (e + p) U \end{bmatrix},$$

$$F = \frac{1}{J} \begin{bmatrix} \rho V \\ \rho u V + \eta_x p \\ \rho v V + \eta_y p \\ (e + p) V \end{bmatrix} \quad \dots\dots\dots (3)$$

ここで、 ρ, u, v , および p はそれぞれ密度, x, y 方向速度成分, 圧力である。単位体積当たりの全エネルギー e は、

$$e = \frac{p}{\gamma - 1} + \frac{\rho}{2} (u^2 + v^2) \quad \dots\dots\dots (4)$$

である。 U, V および J は反変速度成分および座標変換のヤコビアンである。

3.2 数値計算法 式 (2) の空間微分項の離散化には、Yee-Harten の二次精度風上 TVD 法⁽⁵⁾ を用い、時間積分法には、陽的であるが安定条件の緩やかな方法である 2 段階二次精度の有理ルンゲ・クッタ法⁽⁶⁾ を用いる。線の方法に従い、式 (2) の空間微分項を風上 TVD 法により離散化したとき、式 (2) は各格子点において次のような時間に対する常微分方程式となる。

$$\left. \frac{dq}{dt} \right|_{i,j} = W(q)|_{i,j} \quad \dots\dots\dots (5)$$

ここで、 $W(q)$ は空間微分項の離散化を、 i および j は格子点番号を表している。式 (5) の時間積分に適用される 2 段階有理解ルンゲ・クッタ法は以下の手順により

Fig. 1 Communication Time

計算を行う。

$$\left. \begin{aligned} \vec{g}_1 &= \Delta t \vec{W}(q^n), \quad \vec{g}_2 = \Delta t \vec{W}(q^n + c_2 g_1) \\ \vec{g}_3 &= b_1 \vec{g}_1 + b_2 \vec{g}_2 \\ \vec{q}^{n+1} &= \vec{q}^n + \frac{2\vec{g}_1(\vec{g}_1, \vec{g}_3) - \vec{g}_3(\vec{g}_1, \vec{g}_1)}{(\vec{g}_3, \vec{g}_3)} \end{aligned} \right\} \dots\dots (6)$$

ここで、 $(\vec{g}_1, \vec{g}_3)$ はベクトル $\vec{g}_1$ と $\vec{g}_3$ の内積を表し、 n は時間段階を表す。パラメータ b_1, b_2, c_2 はそれぞれ以下の値を用いる。

$$b_1=2, \quad b_2=-1, \quad c_2=0.5 \quad \dots\dots\dots (7)$$

さらに収束加速法として、局所時間刻み幅法および陰的残差平均法を適用する。

4. 並列計算法

4.1 領域分割法 並列計算を行うための手法として領域分割法を用いる。領域分割法とは、計算領域において格子をノードプロセッサと同数の小領域に分割し、各プロセッサが分割された小領域の格子上的計算を受けもつ方法である。各小領域の分割された部分は互いに1格子点の重複部分をもっている。この方法を用いて並列計算を行う場合、プロセッサ間でこの重複部分の格子点上の密度などの物理量を1時間段階進むごとにノードプロセッサ間で通信していくことが必要となる。

二つのノードプロセッサで計算を行う場合の標準的なフローチャートを図2に示す。図2では、ホストプロセッサを設けて2台のノードプロセッサの動作を管理している。ホストプロセッサでは、ノードプロセッサにおいて計算を行うための初期設定および管理を行う。すなわち、ノードプロセッサでの計算プロセスの立ち上げ、計算パラメータの入力、計算状況の画面表示および収束判定を行う。各ノードプロセッサではオイラー方程式の計算用ノードプログラムが起動され、ホストプロセッサより計算条件などのデータを受信し、計算を開始する。計算中は境界条件の設定を行うたびに、各小領域の重複部分のデータを該当するノードプロセッサと送受信し、領域分割された境界のデータの受け渡しを行う。1時間段階進むと各ノードプロセッサの残差がホストプロセッサに送信され、ホストプロセッサが全体の計算が収束したか判断し、計算を継続するか、終了するかを各ノードプロセッサに通知する。

4.2 領域分割動的負荷分散法 4.1節で述べた領域分割法に加えて領域分割動的負荷分散法を適用する収束解を求めるときには情報の伝ばする速度が場所により異なるために、局所的に収束の早い部分と遅い部分とが存在する場合がある。領域分割法を用いて並列計算を行う場合、それぞれのプロセッサで受けもつ小領域が早く収束してしまえば、その後その小領域において計算を続けることは非効率的となる。そこで有効にCPU資源を利用するために、早く収束した小領域を受けもつプロセッサでは計算をいったん停止し、収束が遅いプロセッサの小領域を新たに受けもちながら計算を行う方法を採案する。このように計算領域を収束の状態に合わせて分割し、動的に負荷を分散しながら収束の遅い小領域を集团的に計算することにより、全体としての収束を加速し計算量の軽減から計算時間の短縮を図ることができるのではないかと考えられる。

上記の考えに基づき、以下に領域再分割動的負荷分散法と領域再配分動的負荷分散法と名付けた2種の方法についてアルゴリズムを示す。

まず、領域再分割動的負荷分散法の計算アルゴリズムを二つの計算速度の等しいノードプロセッサ PE₁, PE₂ を用いる場合について述べる。この手法は、収束判定基準を満たしたときにそのノードプロセッサの受けもっていた小領域上の計算を一時停止し、まだ収束判定条件を満たしていないノードプロセッサの計算格子を再分割して、各ノードプロセッサに割り当てるという方法である。以下がその計算アルゴリズムである。

Fig. 2 Flowchart for 2PEs system

Fig. 3 Re domain decomposition for dynamic load balancing

Fig. 4 Flowchart of Re-domain decomposition

ただし、図3のように全計算領域を D 、全体の L_2 -残差を Res とし、領域 D より分割された計算領域 D_1 、 D_2 での計算をそれぞれ PE_1 、 PE_2 が受けもつものとする。計算領域 D_1 、 D_2 での L_2 -残差はそれぞれ Res_1 、 Res_2 とする。また、残差の正規化には Res_{max} として現時間段階までの最大の L_2 -残差を、 k_{max} 段階までの収束判定基準として $a_1 > a_2 > \dots > a_{k_{max}}$ を用いる。

(1) 全計算領域 D を PE_1 、 PE_2 にそれぞれ等分な小領域 D_1 、 D_2 を与える。

(2) D_1 、 D_2 において通信を行いながら計算を進めていく。

(3) $Res_1/Res_{max} < a_k$ ならば、 $D_1 = D_{2left}$ 、 $D_2 = D_{2right}$ とする。または、 $Res_2/Res_{max} < a_k$ ならば、 $D_1 = D_{1left}$ 、 $D_2 = D_{1right}$ とする。

(4) 新たに割当てられた小領域で通信を行いながら計算を進めていく。

(5) $Res/Res_{max} < a_k$ となるまで(2)から(4)を繰り返す。 $Res/Res_{max} < a_k$ ならば $k \leftarrow k+1$ とし、もう一度、 $D_1 = D_{left}$ 、 $D_2 = D_{right}$ とする。

(6) (2)から(5)まで $a_{k_{max}}$ まで繰り返す。

ただし、本研究においては全体の計算の収束判定条件として、 $a_{k_{max}} = 1.0 \times 10^{-4}$ とする。

超音速問題では $k_{max} = 1$ として、収束の早い小領域では再度計算を行わず、収束判定条件を満たした小領域から順次計算を終了するものとする。また、遷音速問題では各小領域における収束基準を多段階にして全体として早く収束させるために $k_{max} = 4$ とする。さらに、途中段階の収束判定基準を

$$a_1 = 1.0 \times 10^{-1}, a_2 = 1.0 \times 10^{-2},$$

Fig. 5 Re-distribution of subdomains

$$a_3 = 1.0 \times 10^{-3}$$

とする。以上の計算手順について、 D_1 を受けもつプロセッサの計算アルゴリズムのフローチャートで図4に示す。

次に領域再配分動的負荷分散法を2台のノードプロセッサを用いる場合について述べる。この方法では、図5の上部分に示すようにあらかじめノードプロセッサ数より多数に等分割した計算格子を用いる。この分割された小さな計算格子をタイルと呼ぶ。計算手順としては各タイルの L_2 -残差を求め、領域再分割動的負荷分散法と同様の手法により、収束判定を行い、計算を一時停止するタイルを判定する。計算を一時停止す

るタイルが発生した場合は各ノードプロセッサの受け持つ負荷が等しくなるように残りのタイルをノードプロセッサに再配分する。図5の下部分に領域が再配分されたときの各ノードプロセッサの受け持つタイルを示す。ここで、図5の網掛けの部分は計算を一時停止しているタイルを表している。全タイルがその収束判定基準を満たした場合は、全タイルを再配分して最後の収束判定基準を満たすまで繰返し上記の手続きで計算を行う。基本的なアルゴリズムは領域再分割負荷分散手法と同じであるためフローチャートは示さないが、領域再分割負荷分散手法と異なる点は負荷の分散をタイルを単位として行うところである。

将来、共有メモリー形の並列計算機に適用する場合、スレッドをタイルごとの計算を行うように割当て、収束したタイルを受け持つスレッドの実行を停止し、各プロセッサ間で負荷が等しくなるようなスレッド数に再配分することにより、本手法を容易に利用することが可能となると考えられる。

5. 領域再分割動的負荷分散法の適用

前述した領域再分割動的負荷分散法を用いて、検定問題として流入マッハ数 $M=2.0$ の超音波ダクト内流れと、流入マッハ数 $M=0.675$ の遷音速ダクト内流れの計算を1台から4台のノードプロセッサにより行い、本手法による計算量の軽減の効果について検討する計

Fig. 6 Computational domain

Fig. 7 Convergence history for supersonic flows

算領域は図6に示すような形状をしている。

5.1 超音波ダクト内流れの計算結果 格子点数 121×41 における本計算法の収束履歴を図7に示す。本並列計算手法による収束解が得られていることが確認できる。図8に各プロセッサ数で計算した収束解の等マッハ線図を本手法を用いずに計算した結果と重ねて示すが、得られた収束解は完全に一致しており、この流れ場に対して本手法の与える収束解への影響はないことがわかる。前述したようにそれぞれの小領域での収束基準は、 $a_{k=k_{\max}}=1.0 \times 10^{-4}$ とし、 $k_{\max}=1$ としている。図9にPE数とWork数の関係を示す。ここで、Work数とは計算領域の全格子点数に対する計算を行う格子点の比である。これによって $k_{\max}=4$ で計算を行うとWork数は $k_{\max}=1$ のときのように少なくなることがわかる。それは、超音速問題では風上側からしか情報が伝わらないために流入側から順に収束していくためである。4台のノードプロセッサを用いて初期の計算領域を4等分する場合、収束するまでの計算量の最大33%の減少ができています。本手法はノードプロセッサの数に依存して領域が分割されるためにノードプロセッサの数が変わると計算を停止する小領域の分割方法も変わってくる。したがって、ノードプロセッサが1台のときには領域の再分割は行われず、収束解が得られるまで全計算領域で計算を行っている。

5.2 遷音速ダクト内流れの計算結果 格子点数 129×33 における本並列計算手法の収束履歴を図10

Fig. 8 Iso Mach lines for supersonic flows

Fig. 9 Number of Works for supersonic flows

に示す。本計算の結果、各ノードプロセッサ数において収束解が得られていることがわかる。図 11 に各ノードプロセッサ数で計算した収束解の等マッハ線図を超音速流れと同様に本手法を用いずに得られた解と重ねて示すが、得られた収束解は完全に一致しており、遷音速流れに対しても本手法の与える収束解への影響はないことが確認できる。図 12 に収束するまでの反復回数を示す。それぞれの小領域での収束基準を 4 段階にして計算を行っている。それは、遷音速問題では、流れの情報の伝ばはすべての方向に向かっているため、全体にじょう乱が伝わり、徐々に収束解に近づくからである。 $k_{\max}=1$ としても計算を行ったが図 12 からわかるように計算量を大きく減少させることができなかった。

Fig. 10 Convergence history for transonic flows

Fig. 11 Iso-Mach lines for transonic flows

Fig. 12 Number of Works for transonic flows

た。 $k_{\max}=4$ としたときは遷音速流れにおいても収束解が得られるまでの計算量を 4 個のノードプロセッサを用いて最大 29% 減少させることができた。

本計算では少ない格子点数で本計算法を行ったので計算量を 30% 程度しか短縮させることができなかったが、さらに大規模な計算を行うとさらに計算量を短縮することが可能であると期待できる。

6. 領域再配分動的負荷分散法の適用

次にノードプロセッサ数に依存せず、より一般的に計算量の減少を期待できる領域再配分動的負荷分散法を、検定問題として流入マッハ数 $M=2.0$ の超音速ダクト内流れおよび二重だ円を過ぎる $M=8.15$ の超音速流れに対して適用し、本手法による計算量の軽減の効果について検討する。遷音速流れについても本手法の適用を行った収束加速の効果が得られなかったため、超音速流れについてのみの考察とする。

6.1 超音速ダクト内流れ 5.1 節と計算条件について超音速ダクト内流れに対して領域再配分動的負荷分散法を適用する。計算格子は 241×41 とする。領域は流路方向と断面の方向にそれぞれ 12×4 , 8×2 分割する場合と比較のために分割を行わない場合である 1×1 について行っている。このときに得られたワーク数と計算時間を表 1 に示す。ただし、この計算では並列計算は行っていない。ワーク数については 1×1 に対して 8×2 としたときは 44% の計算量を減少させ、33% の計算時間の短縮となっている。 12×4 では 48% の計算量の減少を得ているが、計算時間は 29% の短縮にとどまり、 8×2 と比べて逆に計算時間は増加している。これは、領域の細分化のために計算コード中に無駄な処理を行う必要が生じたためである。次に小領域への分割が 12×4 の場合を並列計算機に適用し、動的負荷分散を行う。表 2 に計算時間と加速比を通常の領

Table 1 Number of works and CPU time

Domain	Work	CPU Time (s)
1x1	1466	5981
8x2	817	3982
12x4	760	4225

Table 2 Number of works and CPU time

Nodes	Dynamic Load Balancing		Static Load Balancing	
	CPU Time(s)	Speedup	CPU Time(s)	Speedup
1	4225	1.00	5043	1.00
2	2412	1.75	2725	1.85
4	1663	2.54	1681	3.00

Fig. 13 Geometry of double ellipse

域分割により計算を行った場合とともに示す。ここで加速比はそれぞれの方法の1台のノードプロセッサで計算を行った場合との計算速度の比を表し、理想的にはノードプロセッサ数に正比例する。ノードプロセッサが1台の場合では計算量の減少により本手法のほうが大幅に計算時間が短縮されていることがわかるが、4台の場合について比較すると通常の領域分割に比べ計算時間の短縮は少ない。これは、加速比を見るとより明らかである。この理由は、本計算で用いた計算機は分散メモリー形であり、領域の再配分が行われるたびにその小領域の計算データを通信する必要があるからである。また、小領域の単位で負荷分散を行うために、ノードプロセッサの数に対し十分に小領域の数が多くなければ等しく負荷を分散できないためである。

6・2 二重だ円を過ぎる超音速流れ ダクト内は流れの方向に対して沿う格子を作成することができるが、一般的な形状に対しては物体に沿って格子を作成する。このときの本手法の計算量軽減効果を調べるため二重だ円で構成される飛行機頭部を模した形状まわりの超音速流れに適用する。図13に形状の概略を示す。計算条件は一樣流マッハ数 $M=8.15$ 、迎え角 $\alpha=30^\circ$ である。計算格子については格子点数は 257×97 であり、図14のように 8×3 の小領域に分割する。本計算は1台の計算機で行っている。表3に収束するまでのワーク数と計算時間を分割を行わずに計算した場合とともに示す。これより、本手法によってワーク数が46%減少し、計算時間で19%短縮していることがわかり、領域再配分動的負荷分散法が、実際の形状に対しても有効であることが確認できる。

Table 3 Number of works and CPU time

Domain	Work	CPU Time (s)
1x1	4018	16915
8x3	2156	13705

Fig. 14 Computational grid for double ellipse

7. 結 言

領域再分割動的負荷分散法および領域再配分動的負荷分散法を提案し、オイラー方程式の並列計算をPVMにより行い、以下の結果を得た。

- (1) 領域再分割動的負荷分散法を超音速二次元ダクト内流れに適用し並列計算を行った結果、4台のノードプロセッサを用いたときに計算量が最大33%減少した。
- (2) 領域再分割動的負荷分散法を遷音速二次元ダクト内流れに適用し並列計算を行った結果、4台のノードプロセッサを用いたときに計算量が最大29%減少した。
- (3) 収束加速効果がノードプロセッサの数に依存しない領域再配分動的負荷分散法を超音速二次元ダクト内流れに適用し、計算量が最大48%減少し、計算時間で最大33%短縮した。
- (4) 領域再配分動的負荷分散法を二重だ円を過ぎる超音速流れに適用し、流れに沿わない形状の格子においても領域を 8×3 の小領域に分割した場合、計算

量が46%減少し、計算時間で19%短縮した。

文 献

- (1) Alonso, J. J., ほか3名, *Proc. Parallel CFD'94*, (1994), 289-291.
 - (2) Satofuka, N., Obata, M. and Suzuki, T., *Proc. Parallel CFD'93*, (1995), 525-532.
 - (3) Vidwans, A. and Kallideris Y., *AIAA 93-3313 CP* (1993).
 - (4) Geist, A., ほか5名, *PVM3 User's Guide and Reference Manual*, (1993).
 - (5) Yee, H. C., *NASA TM 89464* (1987).
 - (6) Wambecq, A., *Computing*, **20**(1978), 333-342.
-