清水建設研究報告
第69号平成11年4月

並列処理による流れ場のシミュレーション

三浦　靖弘
(技術研究所)

Numerical Simulation of Flow Fields using Parallel Processing

by **Nobuhiro Miura**

Abstract

The purpose of this paper is to apply parallel processing techniques, which have characteristics such as low cost and high performance, to numerical simulations of flow fields. From within the various parallel processing methods, techniques based on the MIMD and distributed memory architecture are dealt with in this paper since these techniques can be applied not only to parallel machines but also to a network cluster of low-priced personal computers. Also algorithms for reducing overheads at data transfer between some processors and improving convergence are studied. Finally the results of some cases are introduced.

概　要

低コストかつ高速演算性能を兼ね備える並列処理方式を流れ場のシミュレーションに適用することを検討する。並列処理方式には様々な形態が存在するが、本報では並列計算機のみでなく安価な並列処理環境であるネットワークで繋がれたパーソナルコンピュータ群も利用可能なMIMD(Multiple Instruction Multiple Data)、分散メモリアーキテクチャに基づく並列処理方式に関し報告する。また、並列処理において問題となる通信時間等のオーバーヘッドを低減し収束性を向上させるアルゴリズムや並列処理の特徴を生かした大規模計算の例を紹介する。

§１．はじめに

設計および施工の各フェーズにおいて温熱気流環境シミュレーションの要望は非常に多く、設計検討用としてのみでなくプレゼンテーション用として期待が大きい。反面、計算コストや時間的な制約が問題となっている。そこで、本報では低コストかつ高速演算性能を兼ね備えると期待される並列処理方式を流れ場のシミュレーションに適用することを検討する。

並列計算機もしくはネットワークで繋がれたパーソナルコンピュータ群等を使用し高速演算処理を実現する上で重要な課題は、プログラムの移植性と高い並列計算効率のあるソフトウエアの開発である。しかし、並列計算機ハード自身は開発途上の技術で、過去様々な形態のハードが開発されたがその寿命は短く、対応するソフトウェア開発環境もそれぞれのハードの効率優先のため各並列計算機ごとに行われ、開発されたプログラムの相互互換性（移植性）はほとんど無かった。しかし1990年代に入って、KSR-1など例外的に共有メモリを実現させた例はあるが、大規模並列計算（100以上のプロセッサによる並列計算）では、MIMD (Multiple Instruction Multiple Data)、分散メモリアーキテクチャに基づくハードに収束する傾向が顕著となり、これに合わせてソフトウエア開発環境の標準化が進められ、その中でもMPI（Message Passing Interface）は現状および将来的にほとんどの並列計算機メーカーがサポートする予定である。

本報では、並列計算機のみでなく安価な並列処理環境であるネットワークで繋がれたパーソナルコンピュータ群も利用可能なMPIを中心に、現在の並列処理ソフトウエア開発環境の概要を述べ、メッセージパッシングサブセットと既存のプログラム言語により並列計算処理を実現する方法について説明を加える。また、分散メモリアーキテクチャに基づく並列処理において避けられない各プロセッサ間のデータ交換による通信のオーバーヘッドを低減し、さらに計算の収束性を向上させるアルゴリズムに関し報告する。最後に、このMPIを用いた非定常大規模問題の解析例として、チャンネル乱流のダイレクトシミュレーションを実施したのでその結果を報告する。

§２．並列処理ソフトウエア開発環境

並列処理プログラムは、①並列プログラミング言語、もしくは②逐次プログラミング言語＋メッセージ・パ

ッシング・ライブラリの形で記述されるのが主流である。この他、各計算機メーカーがサポートする並列数値計算ライブラリを、逐次プログラムからコールすることにより並列処理を実現する形態もある。並列処理プログラムの作成は、並列計算機以外の逐次処理計算機等への移植性や現状で並列処理プログラムのデバッキングが容易でないことから、まず逐次処理計算機で実行可能なプログラムを開発し、その後並列化する過程が推奨されている。

2.1 並列プログラミング言語

並列プログラミング言語は、逐次言語で書かれた既存のプログラム資産の活用、プログラミングの容易性という長所を持つ反面、言語の文法により並列化のモデルが限定されるという短所を持つ。また、すべての計算機メーカー、ソフトウェアベンダーがサポートするわけではないので移植性に関してはリスクがあり、さらにネットワーク並列処理環境では使用できない。並列プログラミング言語は、元々 SIMD (Single Instruction Multiple Data) アーキテクチャの並列計算機で使われるデータ並列処理の考えに基づいており、HPF (High Performance Fortran) [1]、C*(C-Star)等がある。HPF プログラムでは、Fortran 90 のコメント行 (*HPF$で始まるディレクティブ行) の形でデータの配置、分散等を記述し、Do ループの並列化を行う。HPF プログラムは通常の逐次処理計算機でも再コンパイルすることにより実行が可能で、ベクトル計算機等様々な計算機を使用する可能性のあるユーザーには便利となる。

2.2 メッセージ・パッシング・ライブラリ

メッセージパッシングでは、従来の Fortran (現状では主に Fortran77)、C 等の逐次言語によるプログラムに、並列処理で必要となるプロセッサ間の通信という低レベルな動作を直接記述して並列処理を実現する。HPF などと異なり、開発されたプログラムを通常のベクトル計算機等、並列計算機以外で使用するには、プログラム中に多少の工夫が必要となる。メッセージ・パッシングは様々な並列処理方式が記述できるという長所を持つ反面、プログラミングが難しくなるという短所を持つ。主なメッセージ・パッシング・ライブラリとして、分散および共有メモリ型並列計算機を対象とする PVM [2]、MPI [3]や共有メモリ型並列計算機を対象とする Linda 等がある。PVM (Parallel Virtual Machine)は、Tennessee 大学と Oak Ridge 国立研究所で開発されたもので、並列計算機やネットワークでつながれた異機種計算機群 (ネットワーク・クラスタ) 上で実行可能である。PVM を実行すると、各計算機ノードで pvmd というデーモンが起動され、これがプロセッサ間の通信とプロセス(タスク)管理等を行う。MPI (Message Passing Interface)は、PVM と同様にこのメッセージ・パッシングのインターフェースの標準化を計ったもので、1992 年 11 月に欧米の大学、公的研究機関、IBM を始めとするコンピュータ・メーカーやソフトウェア・ベンダー等が参加して MPI Forum を設立して作業が進められ、1995 年 7 月にはほぼ最終仕様が公表された。ただし現状の MPI は、異機種計算機群で動作することは保証されていない。PVM は多くの計算機環境で動作するが、基本的に各計算機用に最適化されたものでないため、通信のオーバーヘッド等が発生する可能性が考えられるが、MPI の場合はインターフェースが標準化され、各計算機メーカーが独自の最適化されたメッセージ・パッシング・ライブラリのインターフェースを MPI 仕様に合わせるため、処理時間に関して有利である。

§3. MPIによるプログラミング法

MPI ライブラリには 100 を越えるサブルーチンが存在するが、その中で領域分割による流れ場の解析を行う一般ユーザーが使用すると思われるのは 40 程度、特に頻繁に使用するのは 10 程度である。ここでは、プログラミングの基本となる環境管理、メッセージ通信、派生データについて詳述する。MPI ライブラリは FORTRAN および C から利用可能であるが、ここでは FORTRAN を例に取る。

3.1 SPMDモード

MPI は SPMD (Single Program Multiple Data) モードでの動作が推奨されている。これは、すべてのプロセッサで同一プログラムを実行するモードである。このため SPMD によるプログラムは、並列計算の全体を統括するマスター部とマスターの指示により並列処理を行うスレイブ部分の両者が同一プログラムに記述される。

3.2 環境管理

MPI を使用する上で必須となるのが MPI プロセス環境管理である。最も基本となるものは、①MPI が使用する内部テーブル類の定義および②MPI プロセスの起動と③終了で、以下にメインプログラムの基本形態を示す。

```
PROGRAM  MAIN
INCLUDE(MPIF.H)                    ①
```

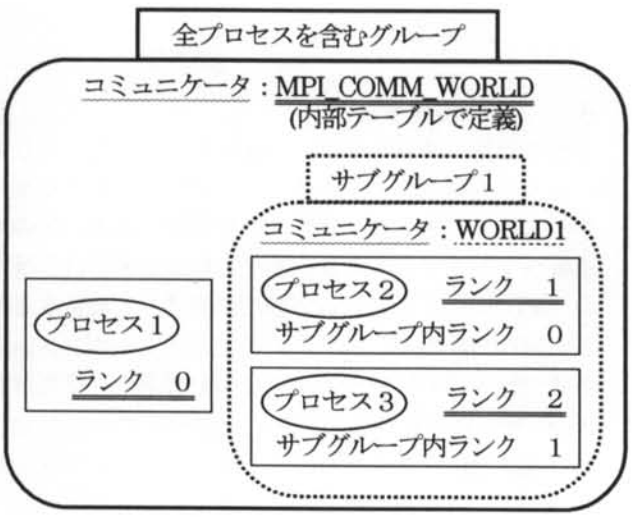
```
      CALL   MPI_INIT(IERR)                    ②
           (計算処理ルーチン)
      CALL   MPI_FINALIZE(IERR)                ③
      END
```

ここで、引数 IERR にはサブルーチンの終了状態が戻り値として代入される。次に、計算処理ルーチンの初期に行うものとして、④プロセスの集合（グループ）内のプロセス総数（変数 NPROCS）の把握と⑤自身のプロセスの識別子（ランク：変数 MYRANK）の取得がある。

```
      CALL MPI_COMM_SIZE(
     &   MPI_COMM_WORLD,NPROCS,IERR)   ④

      CALL MPI_COMM_RANK(
     &   MPI_COMM_WORLD,MYRANK,IERR)   ⑤
```

なお MPI では、プロセス総数は実行時の利用可能な総プロセッサ数を与えるもので、実行途中でプロセッサ数を増すことは出来ない。プロセス識別子（ランク）は0から始まる整数値が与えられ、一般にランク0のプロセスをマスター、それ以外のものをスレーブとして並列処理を行う。ここで、引数 MPI_ COMM_ WORLD は MPI の内部テーブル定義ファイル MPIF.H で定義されるもので、全並列処理プロセスをハンドルするためのグループ名（コミュニケータ）である。MPI では、並列処理グループ内の任意のプロセスをサブグループとしてハンドルするためのサブグループ用コミュニケータを、サブルーチン MPI_COMM_SPLIT( )を用いて定義できる。これにより、例えばサブグループ用に得られたコミュニケータ WORLD1 を⑤のランク取得用ルーチンで MPI_ COMM_ WORLD の代わりに与えることで、サブグループ内の情報を取得することができる。上記のコミュニケータとランクの概念を**図－1**に示す。図に示すように MPI プログラム内では、例えばプロセス2は｛MPI_COMM_ WORLD、ランク1｝もしくは｛WORLD1、ランク0｝で指定できることになる。このサブグループを用いることによりプロセス数を減じた並列計算処理を行ったり、あるいは例えば流体計算と放射計算を別々のサブグループに割り当て、平行して処理させたいというような場合に使用できる。



図－1 コミュニケータとランクの概念

3．3 メッセージ通信

MPI のメッセージ通信用ライブラリには、大きく2つのタイプがある。グループ通信と一対のプロセス間用である。前者は、1回のサブルーチンコールでグループ内の複数のプロセスが一斉にデータのやり取りを実行でき、また自動的に各プロセスの同期を取るものとなっている。しかしこれは、送受信のために2つの配列を用意する必要があるという短所がある。すなわち、例えばいくつかのプロセスが A という配列を分割して計算し、その結果をお互いがやり取りする場合、相手からデータを受信するためには一端 B という別の配列で受け取り、改めて B から A に代入するという操作が必要となる。このグループ通信は、領域分割型並列流体計算ではほとんど使用することは無いと思われる。MPI は後者の1対1通信サブルーチンを数多く提供しているが、ここでは一般的な領域分割型並列流体計算プログラミングで必要と思われる4つのサブルーチンを紹介する。まず、プロセス間の同期を取りながら実行するタイプのブロッキング送信、受信がある。これらは、送受信処理が完了するのを待って次の処理へ進む。

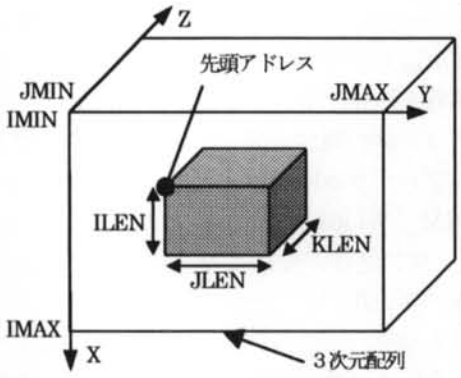
```
      CALL MPI_SEND(BUF,COUNT,DATATYPE,
     &   DEST,TAG,COMM,IERR)

      CALL MPI_RECV(BUF,COUNT,DATATYPE,
     &   SOURCE,TAG,COMM,STATUS,IERR)
```

また、特に同期を取らずに実行するタイプ、非ブロッキング送信および非ブロッキング受信がある。

```
      CALL MPI_ISEND(BUF,COUNT,DATATYPE,
     &   DEST,TAG,COMM,REQUEST,IERR)

      CALL MPI_IRECV(BUF,COUNT,DATATYPE,
     &   SOURCE,TAG,COMM,REQUEST,IERR)
```




```
      SUBROUTINE DATATYPE_CREATE(
     &  IMIN,IMAX,JMIN,JMAX,ILEN,JLEN,KLEN,IOLDTYPE,INEWTYPE)
      INCLUDE(MPIF.H)
      CALL MPI_TYPE_EXTENT(IOLDTYPE,ISIZE,IERR)
      CALL MPI_TYPE_VECTOR(JLEN,ILEN,IMAX,IMIN+1,IOLDTYPE,ITEMP,IERR)
      IDIST=(IMAX-IMIN+1)*(JMAX-JMIN+1)*ISIZE
      CALL MPI_TYPE_HVECTOR(KLEN,1,IDIST,ITEMP,INEWTYPE,IERR)
      CALL MPI_TYPE_COMMIT(INEWTYPE,IERR)
      END
```

図－2　3次元配列中の直方体型派生データと定義用サブプログラム例

ここで、BUFは通信データの先頭アドレス、COUNTは通信データの要素数、DATATYPE は通信データの型（3．4節参照）、DEST は宛先プロセスのランク、SOURCE は送信元プロセスのランク、TAG はユーザーが定義するメッセージの区分、COMM は送受信相手プロセスが属するグループのコミュニケータ、STATUS は MPI の内部テーブルで定義された MPI_STATUS_ SIZE の大きさの整数配列で通信状態のタグが代入される（一般には使用しない）。REQUEST はサブルーチンより返されるメッセージの識別子で、後述する通信処理完了待ちサブルーチンで対象となるメッセージの判別に使用する。送受信を行う場合、これらのサブルーチンのどのような組み合わせも許されるが、ブロッキングタイプを使用する場合は各プロセスがお互いに相手の通信処理完了待ち状態になり動けなくなる、いわゆるデッドロックに気を付ける必要がある。また、非ブロッキングタイプを使用する場合、通信処理が完了する前に次処理を実行するため、送受信用システムバッファを通信完了前に破壊する可能性があり、プログラマーの責任でこれを回避しなければならない。そのために MPI では MPI_WAIT (REQUEST, STATUS, IERR)という通信処理の完了待ちをさせるためのサブルーチンを用意している。REQUEST には対象となる送受信メッセージの識別子を与える。非ブロッキング送信 MPI_ISEND( )＋完了待ち MPI_WAIT( )は、ブロッキング送信 MPI_SEND( )と動作としては全く同じであるが、一般的に非ブロッキング送信＋完了待ちの方が動作が速い。

3．4　派生データ（DERIVED TYPE DATA）

3．3節で示したように、メッセージをやり取りする場合、引数の一つとしてそのメッセージがどのようなデータ型（TYPE）であるのかを指定する必要がある。データ型として MPI_INT（INTEGER 型）、MPI_FLOAT （ FLOAT 型 ）、 MPI_DOUBLE （DOUBLE 型）、MPI_CHAR（CHARACTER 型）等の基本データ型を指定できるが、これらはメモリ上で連続したデータのみを対象としている。領域分割型3次元並列流体計算では、3次元配列中のある断面のデータ（例えば、隣接ブロック境界面の速度 U）をやり取りする必要があるが、このようなデータはメモリ上ではとびとびに配置されており、基本データ型では指定できない。通常の FORTRAN77 や C の使用を念頭とする MPI ではこのような場合の対処として、ユーザー定義のデータ型（派生データ）を作成するためのサブルーチンを提供している。すなわち、ユーザーがある規則で並んでいる不連続データを派生データ型として定義し、これを引数として与えることで不連続データの送受信を可能としている。**図－2**に、3次元配列中の直方体型派生データとそれを定義するためのサブプログラム例を示す。新たに定義された派生データ型が INEWTYPE に返される。サブプログラム内部で使用する4つのサブルーチンの詳細に関してはここでは省略するが、このような非常に簡単なプログラムで直方体型派生データが定義でされる。

§4．　並列処理と収束性

4．1　並列処理手法

流体計算の並列処理は、①離散化された多元連立方程式の並列処理による方法と②物理的流れ場を領域分割して行う方法の大きく分けて2種類がある。①の大規模なマトリックス演算を並列処理する方法は、既存の CFD コードを並列計算機用に改造して解析する際よく用いられる。しかし、この並列化アルゴリズムは各並列計算機のアーキテクチャの影響を大きく受け、HPF 等の効率的な汎用の並列化ツールが未だ完全でない現状では、並列計算効率の良いコードを作成するのは必ずしも容易ではない。また、自動並列化コンパイラも効率の点で改善の余地がある。本研究では、流

体計算の並列処理手法として領域分割法を考察する。

領域分割法では、各ブロック間における格子点の空間的な対応関係が重要となる。コロケーション格子は、速度ベクトル、スカラー(圧力)定義点が同一格子点上で定義されており、各ブロック間での格子データの交換を平易に行うことが可能となる。また、各速度ベクトル成分に関する方程式を共通のコードで記述することができ、またマルチグリッド法などの収束加速法との親和性も高くできる。このような利点を数多く持つことから、本研究では汎用性を考え一般曲線座標系コロケーション格子[4,5,6,7,8]による離散化を行う。

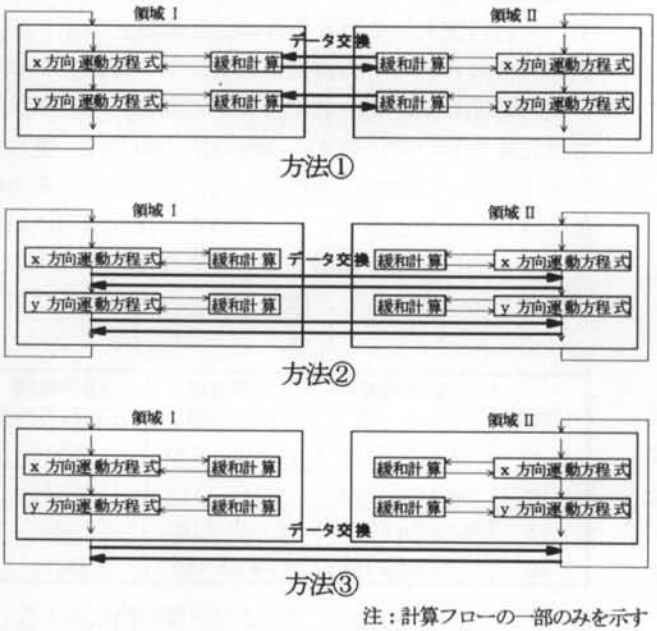
4．2　ブロック境界のデータ交換

領域分割法では、ブロック境界の処理で隣接ブロックの境界データが必要であり、そのデータ交換方法が解の精度や収束性に影響を与える。

ブロック境界におけるデータ交換の方法には、**図－3**に示したように①計算領域全体を一つのマトリックスとみなし、緩和計算の各ステップで必要なデータを交換する方法、②ブロック内の計算は独立して行い各ベクトル、スカラー方程式の緩和計算が終了した段階でデータ交換を行う方法、③各ブロックのすべての緩和計算が終了した段階でデータ交換を行う方法、の３方法が考えられる。現在①の方法が主流となっているが、本計算では、データ交換負荷が軽く通信のオーバーヘッドを低減できることとプログラム作成の簡易性等を考慮し③の方法を採用する。

4．3　収束性の検討

解析領域を１つのブロックとして計算する場合と領域を複数のブロックに分割し、各ブロックごとに独立して緩和計算（1 サイクル）を行い、これが終了した段階で境界データの交換を行う方法では、並列計算の収束性に差が生じ、当然後者は劣るものと考えられる。すなわち、後者の方法は各緩和計算ステップで各ブロックが完全に独立した状態で計算されるため、解析領域全体としての情報伝達が遅くなること、隣接ブロックより別々に受け取った境界値がそのブロック全体の境界での連続式を必ずしも満足していないこと等の収束性悪化の要素が生じる。本研究では、③の方法で収束性を改善する方法として大規模な流体計算における緩和計算加速法であるマルチグリッド手法の概念を適用して、各ブロック毎に厳密に連続条件が満足されるようにブロック境界条件を課すことにより収束性の向上を計ることを検討する。



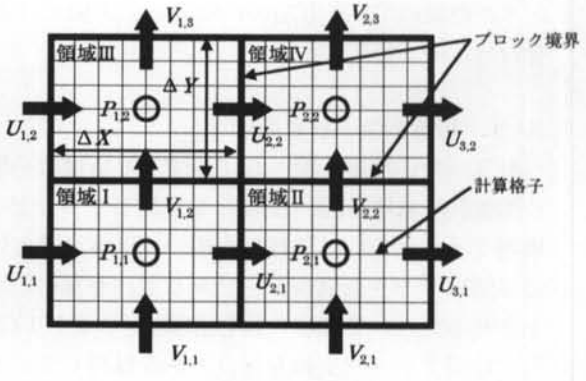
注：計算フローの一部のみを示す

図－3　ブロック間のデータ交換法

4．4　収束加速手法

本報では、緩和計算の過程で(１サイクルごとに)ブロック領域単位に連続式を満たすことを収束緩和計算過程に強制的に課して、収束性の向上を計る。すなわち、**図－4**に示すように（ただし、図は簡単化のため２次元の例を示している)、ブロック境界の計算格子速度の平均値をブロック境界速度（$U_{i,j}, V_{i,j}$)、ブロック内計算格子圧力の平均値をブロック内圧力（$P_{i,j}$)と定義し、各ブロックを一種のセルと見なし各ブロックが連続条件を満足するようブロック境界速度およびブロック内圧力を同時に修正する。本報では、境界速度およびブロック内圧力の修正法として HSMAC 法を適用する。HSMAC 法による圧力修正は、圧力修正による運動量修正を表す次式に基づく。

$$\frac{V^{n+1}-\tilde{V}}{\Delta t}=-\nabla\left(P^{n+1}-P^{n}\right) \qquad (1)$$



U_{ij}, V_{ij}：ブロック境界速度、P_{ij}：ブロック内圧力

図－4　ブロック境界速度とブロック内圧力の定義

(1)式の発散をとり n+1 時点での速度 V^{n+1} が連続条件を満足する条件から P^{n+1} に対する修正量 δP に関して次式を得る。

$$\nabla^2 \delta P = \nabla \bullet \tilde{V} / \Delta t \qquad (2)$$

(2)式左辺の Laplacian を中心差分で近似し、さらに優対角近似をほどこせば、

$$\delta P = -\omega D \Big/ \left\{ 2\Delta t \left(\frac{1}{\Delta X^2} + \frac{1}{\Delta Y^2} + \frac{1}{\Delta Z^2} \right) \right\} \qquad (3)$$

を得る。ここで、ω は加速係数で $1 \leqq \omega \leqq 2$ の範囲が適当な値である。Dはブロックにおける連続式の残差 ε に等しく、

$$D = \varepsilon = \nabla \bullet \tilde{V} \qquad (4)$$

である。よって、(1)式より速度 V^{n+1} の修正量 δV は次式となる。

$$\delta V = -\Delta t \nabla \delta P \qquad (5)$$

境界の速度修正値は以下のようになる。

$$\delta u_{i\pm 1/2, j, k} = \pm \Delta t \delta P / \Delta X \qquad (6)$$

$$\delta v_{i, j\pm 1/2, k} = \pm \Delta t \delta P / \Delta Y \qquad (7)$$

$$\delta w_{i, j, k\pm 1/2} = \pm \Delta t \delta P / \Delta Z \qquad (8)$$

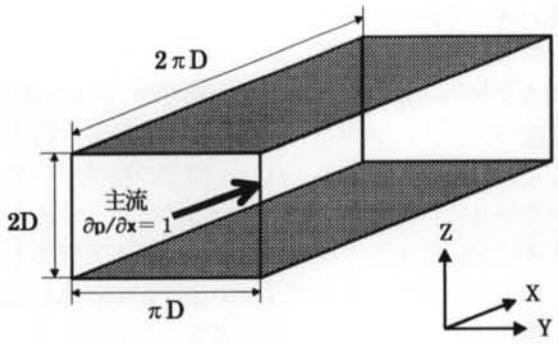
これより求められた速度から新しいDを計算し、(3)式と(5)式を必要回数繰り返し用いることにより連続の式を満足させる。

§5. 解析例

本報では、非定常大規模解問題の解析例としてチャンネル乱流の DNS (Direct Numerical Simulation) を行ったので報告する。

5.1 計算対象と計算条件

計算対象の概要を**図－5**に示す。計算機は分散メモリ型並列計算機 SP2 (IBM) を使用し、メッセージ交換用ライブラリには MPI を用いた。本計算では分割領域間のデータ通信負荷を低減し並列計算の粒度を向上させるため、各領域の緩和計算(1サイクル)が終了した段階でデータ交換を行う方法を採用している。離散化は時間前進には4段階ルンゲ・クッタ法、空間には2次の中心差分法を用いている。並列計算による



図－5 計算対象の概要

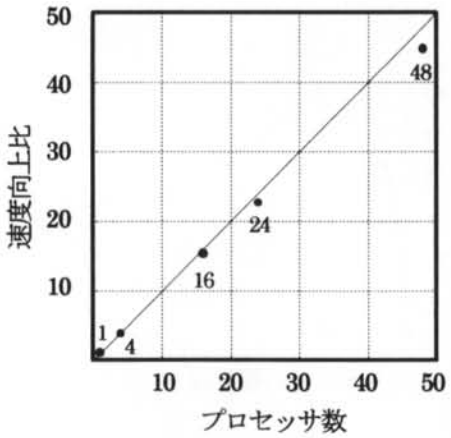
DNS はすべて倍精度計算を行っている。計算条件は、Re_τ数（代表速度：壁面摩擦速度、代表長さ：チャンネル半幅）＝100、主流方向に一定の圧力勾配（$\partial p/\partial x = 1$）を与えている。境界条件は、流れ方向（x方向)、スパン方向（y方向）には周期境界条件、壁面では no-slip 条件を課した。計算領域は $2\pi \times \pi \times 2$、ブロックはx y断面およびy z断面で均等分割した。メッシュはx、y方向は均等分割、z方向は壁面第1メッシュで y^+ がおよそ 1 となるよう分割した。速度向上比の検討は、ブロック分割数を 4、16、24、48 と設定し、最終結果は 24 分割で計算した。また、本報で示した収束加速手法は、同形状のチャンネル流れを対象とし Re＝40000、メッシュ分割は 30×20×20、ブロック分割は 4、k－ε型モデルを基礎とする定常解法 (SIMPLE) に関して検討した。

5.2 並列処理効率

4段階ルンゲ・クッタ法は陽的な時間積分法のため、速度に関する時間積分に関して並列処理に伴う通信負荷はそれほど大きくはならない。圧力修正のポアソン方程式の緩和過程での通信処理負荷がシミュレーション実行時間の効率を左右することになる。今回の計算では正味計算時間に対する通信時間の割合は、**表－1**に示すように分割数の増加とともに増加するがその値は小さく、圧力修正ポアソン方程式の緩和過程においてデータ通信負荷を低減するデータ交換法の採用の効果が確認できる。また、**図－6**にプロセッサ数に対す

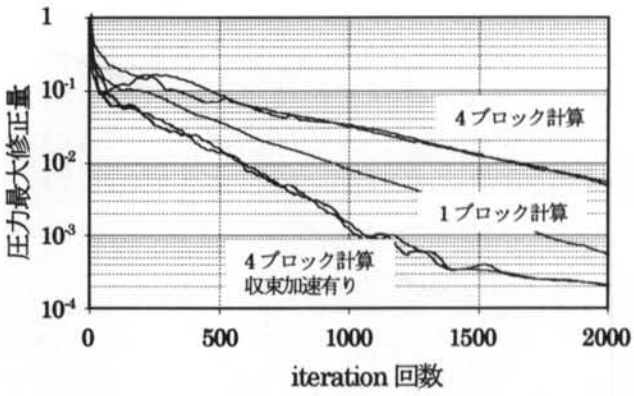
分割数	全体領域 メッシュ数	分割領域 メッシュ数	通信時間 正味計算時間
4	54×54×102	28×28×102	0.039
16	54×54×102	15×15×102	0.046
24	54×56×102	15×11×102	0.06
48	56×58×102	11×9×102	0.071

表－1 ブロック分割数による正味計算時間に対する通信時間の変化



注：速度向上比＝プロセッサ１個の時の処理時間／処理時間

図－６　プロセッサ数と速度向上比



図－７　収束過程

る速度向上比を示す。プロセッサ数に対してほぼリニアな関係となっており、48 プロセッサによる並列処理を実行した場合でも約 45 倍という高い速度向上比が得られた。

５．３　収束加速

図－７に 4 ブロック計算でブロックごとに連続条件を課す収束加速アルゴリズムを導入したものと導入していないものおよび 1 ブロック計算の収束過程をそれぞれ示す。縦軸に計算領域内における圧力修正量の最大値（4 ブロックの場合はブロックごとに示す）、横軸に iteration 回数を示す。図に示されるように、ブロックごとに連続条件を課す収束加速アルゴリズムを導入しない 4 ブロック計算は 1 ブロック計算と比較して、iteration 回数が２倍近く増加し収束性が悪くなっている。これに対し収束加速アルゴリズムを導入したものは、明らかに収束が加速されている。

５．４　計算結果と考察

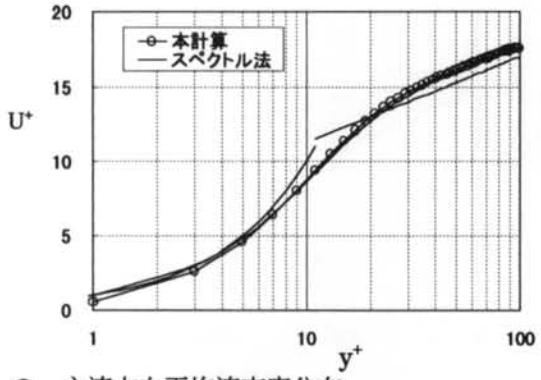
図－８に主流方向平均流速度分布を示す。既存のスペクトル法による結果（黒田ら）と比較している。スペクトル法に比べ精度が劣る差分法で、また壁面第１メッシュが比較的荒い格子を用いたにも関わらず、平均流は非常に良い一致が見られる。**図－９～１１**に主流方向、スパン方向、壁面垂直方向各成分の乱れ強さを、**図－１２**に乱流せん断応力を、スペクトル法による結果と比較して示す。乱れ強さに関しては、全体的に僅かながらではあるが本計算の方が大きめに算出されている。また、乱流せん断応力に関しては、壁面近くでスペクトル法より若干小さくなっている。しかしながら、速度分布と同様に非常に良く一致している。**図－１３**に乱流エネルギーの各項 [9,10]を示す。各項の分布はスペクトル法と同様の性状を示したが、乱流統計量を算出するにあたり運動方程式の差分式との整合性を考慮したコンシステントスキーム [11,12]を用いたにもかかわらず、現状では壁面近くで若干の収支残差が生じている。このことからスペクトル法との差は、壁面近傍のメッシュの粗さと差分近似精度の低さによるものの他に、平均量算出のための平均化時間時間の不足も多少考えられる。

§６．　まとめ

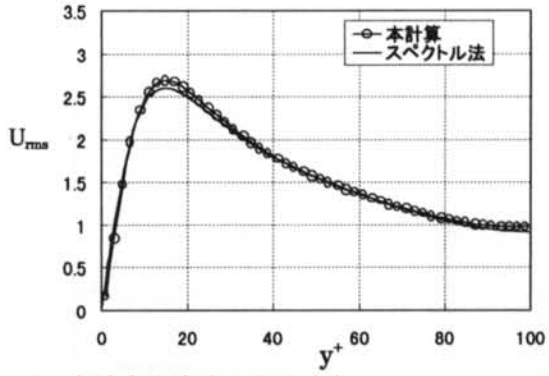
既存の言語にメーセージパッシングサブセットを組み合わせて並列処理を実現する方法は、分散メモリ型のみならず共有メモリ型並列計算機およびネットワークで繋がれたパーソナルコンピューター群における領域分割型並列流体計算に有効となる。なかでも MPI は標準化が計られたシステムで、極めて移植性の高い並列計画処理システムである。MPI を用いた領域分割型並列流体計算の例としてチャンネル流の DNS の結果を示した。計算結果は既存のスペクトル法によるものと比較して、壁面第１メッシュが粗いにも関わらず非常によい一致が見られた。計算速度に関しては、データ交換負荷を低減するアルゴリズムの採用でプロセッサ数にほぼリニアに増加した結果を得た。また、それに伴う計算の収束性の悪化は、ブロックごとに連続条件を課す収束加速アルゴリズムを導入することにより改善できる見通しを得た。

謝辞

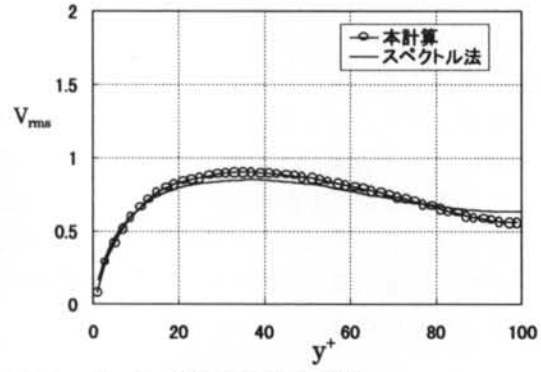
本研究は、東京大学生産技術研究所第５部村上研究室との共同研究として行われたものである。本研究の遂行に当たり村上周三教授、加藤信介助教授には多大なるご助言を賜り、ここに深く感謝の意を表します。



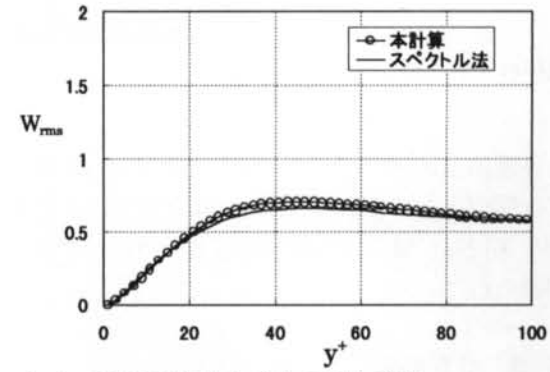
図－８　主流方向平均流束度分布



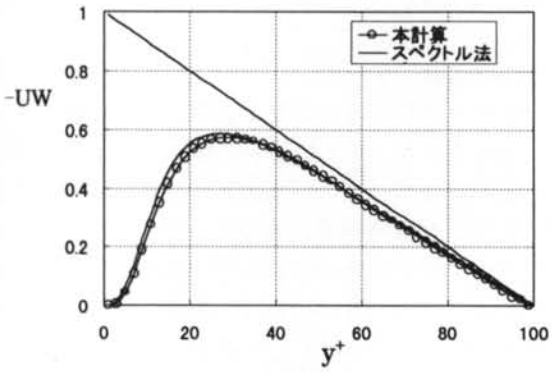
図－９　主流方向成分の乱れ強さ



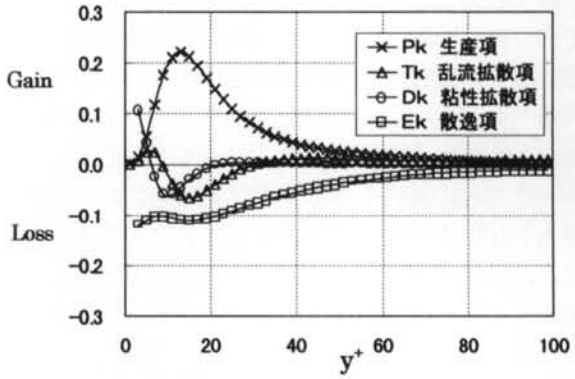
図－１０　スパン方向成分乱れ強さ



図－１１　壁面垂直方向成分の乱れ強さ



図－１２　乱流せん断応力



図－１３　乱流エネルギー k の収支

<参考文献>

1) C.Kelbel, D.Loveman, R.Schreiber, G.Steel, M.Zosel, " The High Performance Fortran Handbook ", Cambridge, MA:MIT Press, 1994
2) A.Geist, A.Beguelin, I.Dongarra, W.Jiang, R.Manchekc and V.Sunderam, " PVM: Parallel Virtual Machine ", Cambridge, MA:MIT Press, 1994
3) W.Gropp, E.Lusk and A.Skjellum, " Using MPI ", Cambridge, MA:MIT Press, 1994
4) F.S. Lien, M.A Leschziner, " A General non-orthogonal Collocated FV Algorithm for Turbulent Flow at all Speeds Incorporating Scond-Moment Closure ", UMIIST, April 1992
5) F.S. Lien, M.A Leschziner, " Computational Modelling of 3D Flow in Complex Ducts and Passages ", UMIST, Feb.1992
6) Rhei, C.M. and Chow, W.L., AIAA journal, 21, pp1525-1532, 1983
7) 村上, 加藤, 石田；"一般曲線座標系による室内気流数値シミュレーション　その３"，日本建築学会計画系論文報告集,第 400 号, 1989
8) 村上, 加藤, 石田；"一般曲線座標系による室内気流数値シミュレーション　その４"，日本建築学会計画系論文報告集,第 408 号, 1990
9) J.Kim, P.Moin and R.Moser, " Turbulence statistics in fully developed channel flow at low Reynolds number ", J.Fluid Mech.(1987), vol.177, pp.133-166
10) N.N.Mansour, J.Kim and P.Moin, " Reynold-stress and dissipation-rate budgets in a turbulent channel flow ", J.Fluid Mech.(1988), vol.194, pp.15-44
11) 梶島；"対流項の差分形式とその保存性"，日本機械学会論文集(B編), 60 巻, 574 号, pp.2058-2063, 1994
12) 鈴木, 河村；"乱流の直接シミュレーションにおける差分式の整合性"，日本機械学会論文集(B編), 60 巻, 578 号, 1994

付録　ＭＰＩを用いたサンプルプログラム（抜粋）

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCC        MPI SAMPLE            CCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      IMPLICIT DOUBLE PRECISION(A-H,O-Z)
      PARAMETER(NX=31 ,NY=31 ,NZ=31)
C
      INCLUDE (mpif.h)                        ;内部テーブル類定義
      INTEGER ISTATUS(MPI_STATUS_SIZE)    ;MPI_WIAT 文のステータス入力配列
                                              通常使用しないが、宣言文は必要
C
      COMMON /VARS/
     1 U(NX,NY,NZ),V(NX,NY,NZ),W(NX,NY,NZ),P(NX,NY,NZ)
C
      CALL MPI_INIT(IERR)    ;MPI プロセス起動
      CALL MPI_COMM_SIZE(MPI_COMM_WORLD,NPROCS,IERR)    ;プロセス総数取得
      CALL MPI_COMM_RANK(MPI_COMM_WORLD,MYRANK,IERR)    ;自プロセス識別子取得
C
      NTASKS=NPROCS    ;総プロセス数
      ID=MYRANK        ;自プロセスの識別子
C
      IBCW=1           ;送受信データの種類指定子
C
C*********************************************************************
C     DIFINE TYPE OF TRANSFER DATA
C*********************************************************************
C
      IMIN=1
      IMAX=NX
      JMIN=1              ;３次元配列の大きさ指定
      JMAX=NY
      KMIN=1
      IOLDTYPE=MPI_DOUBLE_PRECISION    ;元データのデータ型指定（ここでは、倍精度実数型）
C-----WEST   ;west 面のデータ型指定
C             先頭アドレスは(1,1,1)
      ISTART=1
      IEND=1
      JSTART=1
      JEND=NY             ;定義したい派生データの大きさ指定
      KSTART=1
      KEND=NZ
      CALL PARA_TYPE_BLOCK(IMIN,IMAX,JMIN,JMAX,KMIN,ISTART,IEND,
     &                     JSTART,JEND,KSTART,KEND,IOLDTYPE,INEWTYPE)
      ITYPWEST=INEWTYPE    ;定義された派生データ
C-----EAST   ;east 面データ型指定
C             先頭アドレスは(1,1,1)
      ISTART=NI-1
      IEND=NI-1
      CALL PARA_TYPE_BLOCK(IMIN,IMAX,JMIN,JMAX,KMIN,ISTART,IEND,
     &                     JSTART,JEND,KSTART,KEND,IOLDTYPE,INEWTYPE)
      ITYPEAST=INEWTYPE
C*********************************************************************
C
      DO 1000 LOOP=1,MAXITERATION
C
      CALL CALCU
      CALL CALCV
      CALL CALCW
      CALL CALCP
C
```

```
C*********************************************************************
C     TRANSFER B.C DATA   (1境界面のデータ送受信のみ示す。)
C*********************************************************************
C*****WEST  (NX-1,1,1)-->(1,1,1)
      IDEST=ID+1   ;送信先識別子指定
      MSRC=ID-1    ;送信元識別子指定
C
      ICOUN=1      ;送受信データの要素数 (派生データはひとかたまりの要素で1)
      ISTYP=ITYPEAST   ;送信データ型指定 (ここでは、EAST面の派生データ型)
      IRTYP=ITYPWEST   ;受信データ型指定 (ここでは、WEST面の派生データ型)
      ITAG=IBCW        ;送受信データの種類指定
      ICOMM=MPI_COMM_WORLD   ;コミュニケータの指定
C
      CALL MPI_ISEND(U(1,1,1),ICOUN,ISTYP,IDEST,ITAG,ICOMM,ISREQ,IERR)
      CALL MPI_IRECV(U(1,1,1),ICOUN,IRTYP,MSRC,ITAG,ICOMM,IRREQ,IERR)
      CALL MPI_WAIT(ISREQ,ISTATUS,IERR)
      CALL MPI_WAIT(IRREQ,ISTATUS,IERR)
      CALL MPI_ISEND(V(1,1,1),ICOUN,ISTYP,IDEST,ITAG,ICOMM,ISREQ,IERR)
      CALL MPI_IRECV(V(1,1,1),ICOUN,IRTYP,MSRC,ITAG,ICOMM,IRREQ,IERR)
      CALL MPI_WAIT(ISREQ,ISTATUS,IERR)
      CALL MPI_WAIT(IRREQ,ISTATUS,IERR)
      CALL MPI_ISEND(W(1,1,1),ICOUN,ISTYP,IDEST,ITAG,ICOMM,ISREQ,IERR)
      CALL MPI_IRECV(W(1,1,1),ICOUN,IRTYP,MSRC,ITAG,ICOMM,IRREQ,IERR)
      CALL MPI_WAIT(ISREQ,ISTATUS,IERR)
      CALL MPI_WAIT(IRREQ,ISTATUS,IERR)
      CALL MPI_ISEND(P(1,1,1),ICOUN,ISTYP,IDEST,ITAG,ICOMM,ISREQ,IERR)
      CALL MPI_IRECV(P(1,1,1),ICOUN,IRTYP,MSRC,ITAG,ICOMM,IRREQ,IERR)
      CALL MPI_WAIT(ISREQ,ISTATUS,IERR)
      CALL MPI_WAIT(IRREQ,ISTATUS,IERR)
C*********************************************************************
 1000 CONTINUE
C
      CALL MPI_FINALIZE(IERR)   ;MPIプロセスの終了
C
      STOP
      END
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C     派生データ定義サブプログラム (本文中とは別種類のものを示す。)
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      SUBROUTINE PARA_TYPE_BLOCK(IMIN,IMAX,JMIN,JMAX,KMIN,
     &           ISTA,IEND,JSTA,JEND,KSTA,KEND,IOLDTYPE,INEWTYPE)
      IMPLICIT DOUBLE PRECISION(A-H,O-Z)
      INCLUDE (mpif.h)
      INTEGER IBLOCK(2),IDISP(2),ITYPE(2)
      CALL MPI_TYPE_EXTENT(IOLDTYPE,ISIZE,IERR)
      ILEN = IEND - ISTA + 1
      JLEN = JEND - JSTA + 1
      KLEN = KEND - KSTA + 1
      CALL MPI_TYPE_VECTOR(JLEN,ILEN,IMAX-IMIN+1,IOLDTYPE,ITEMP,IERR)
      IDIST = (IMAX-IMIN+1)*(JMAX-JMIN+1)*ISIZE
      CALL MPI_TYPE_HVECTOR(KLEN,1,IDIST,ITEMP,ITEMP2,IERR)
      IBLOCK(1) = 1
      IBLOCK(2) = 1
      IDISP(1) = 0
      IDISP(2) = (  (IMAX-IMIN+1)*(JMAX-JMIN+1)*(KSTA-KMIN)
     &           + (IMAX-IMIN+1)*(JSTA-JMIN) + (ISTA-IMIN)  )*ISIZE
      ITYPE(1) = MPI_LB
      ITYPE(2) = ITEMP2
      CALL MPI_TYPE_STRUCT(2,IBLOCK,IDISP,ITYPE,INEWTYPE,IERR)
      CALL MPI_TYPE_COMMIT(INEWTYPE,IERR)
      RETURN
      END
```