56

並列計算CFDプラットフォームUPACSについて

山本一臣、榎本俊治、高木亮治、山根 敬、山崎裕之、山本 武、藤田直行、松尾裕一、
岩宮敏幸、野崎 理、大西 充、溝渕泰寛、牧田光正、黒滝卓司、藤原 仁志
(航空宇宙技術研究所 UPACS プロジェクトチーム)

On a Parallel CFD Platform - UPACS

by
K. Yamamoto, S. Enomoto, R. Takaki, T. Yamane, H. Yamazaki, T. Yamamoto, N. Fujita, Y. Matsuo,
T. Iwamiya, O. Nozaki, M. Ohnishi, Y. Mizobuchi, M. Makida, T. Kurotaki, H. Fujiwara

ABSTRACT

This paper presents a project of CFD code development, UPACS (**U**nified **P**latform for **A**erospace **C**omputational **S**imulation), started recently in NAL. The project aims firstly to overcome the increasing difficulty in the recent CFD code programming for aerospace applications which includes complex geometry, coupling with heat transfer, structure dynamics and so on as well as parallel computational techniques. Secondly it also aims to accelerate the development of CFD technology by sharing a common base code (platform) among CFD related research scientists and engineers. The basic concept of the code design, the parallel computational method, the multi-block method and its programming using Fortran 90 are briefly explained. The focus is placed on a hierarchical structure of the code to realize a concept of separating the flow solving process from both the complicated multi-block data handling and the data transfer process in the parallel computation without losing flexibility and applicability.

1．はじめに

数値流体力学（CFD）の計算法の進歩と、並列計算機ハードウェアの進歩により、高い計算精度が要求される航空宇宙分野でも複雑な非定常流や化学反応流などの大規模計算、航空機、宇宙往還機や推進機の複雑形状についての流れ解析、構造や伝熱との連成解析による設計の最適化などが可能となってきている。

しかし、このような複雑な計算を行う場合のプログラム開発は容易な事ではない。並列化に関しては、コンパイラによる自動並列化技術はまだ不充分であり、NWT Fortranでは制約が多く、また、MPI[1]やPVM[2]といったメッセージ・パッシング・ライブラリを利用する場合は、ライブラリの知識と並列計算モデルの設計が必要になってくる。また、複雑形状に適応させた計算には、マルチブロック構造格子法と非構造格子法の2種類の方法が考えられるが、マルチブロック構造格子法の場合は、ブロック間のデータ交換手法を汎用化しなければ、ブロック数が10以上にもなると対応できなくなる。さらに、連成計算が必要な場合、数種類の物理モデルをまとめて計算を行おうとするときには、従来の流体計算コードに付加する形では、プログラムは煩雑なものになることがある。

このようなプログラミングの課題は、CFDの基本となる、スキーム、境界条件、乱流モデル、化学反応モデルといった物理モデルやアルゴリズムとは異種のものであり、CFD技術者・研究者にとっては煩雑なプログラミングを強いられ、多大な労力を必要とする。さらに、多くの場合、このような複雑化したプログラムが一般化されておらず、類似の方法を用いたコードが種々の分野で重複して開発されている。このような開発手法は、コストがかかるばかりでなく、系統だったCFDコードの検証やノウハウの蓄積を行う上でも効率的とは言えず、ある種の汎用的な標準プログラミング・スタイルやライブラリを確立する必要があると考えられる。

この状況に対する動きとして、流体計算と並列処理の分離を目的としたコード開発が一部では進められている。例えば、ドイツ航空宇宙センター（DLR）では90年代の前半にターボ機械流れの計算を対象としたTRACE[3]というC言語による並列計算コードを開発しており；日本原子力研究所の太田はC++言語を用いたオブジェクト指向によるフレームワーク[4]を提案している。

同様に、航空宇宙技術研究所でも1998年度から、プログラムの複雑化に対応するため、各分野のCFD研究者が協力して、それぞれのCFDコードに共通するプログラム構造を分析し、分野が異なる研究者間でも共有できる基盤コード（プラットフォーム）を確立することを目指して、UPACS（**U**nified **P**latform for **A**erospace **C**omputational **S**imulation）というプロジェクトを進めている。この基盤コードにより、上で述べたプログラムの複雑化の問題を解決するとともに、スキームや乱流モデルなどのCFDの基盤的な研究から、現象解明の研究や設計といったコードの応用までを実質的にリンクすることが期待でき、それぞれの段階における成果や課題のフィードバックが効率的に行われることを狙って

いる。現在は、まず、その第1段階として、多くの研究者・技術者が苦労している、複雑形状まわりの計算、並列計算、連成問題の計算を容易にすることを課題として、プラットフォームの開発を進めている。ここでは、コード設計の考え方など、開発の鍵になる内容を紹介する。

2．コード設計の方針

UPACSコードの開発方針としては、次のような考え方を取っている。

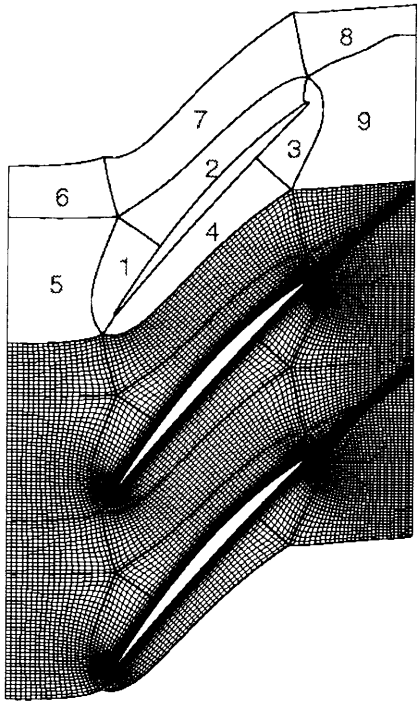
(1) 複雑形状への対応のためにマルチブロック構造格子法を利用する

(2) 並列処理部／マルチブロック処理部と、ソルバー部の明確な分離を行い、ユーザーはシングルブロック・イメージのソルバーを最小限変更することで目的の計算が可能になるようにする

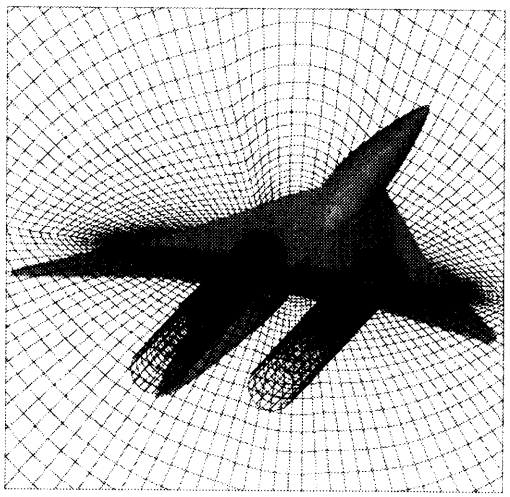
(3) ハードウェアへの依存性を最小限にする（シングルプロセッサー機かマルチプロセッサー機か、スカラー機かベクトル機か、また、分散メモリ型か共有メモリ型かなど）

(4) ソースコードレベルの共有が可能になるように、データおよび副プログラムの階層構造の明確化、モジュールのカプセル化を行う

(1)の方針に関しては、現在、航技研のコードの大半が構造格子法を用いているために、マルチブロック構造格子法を対象にしているが、将来、非構造格子法も導入できるように進めている。(2)(3)の方針のために、並列化にはMPIを利用し、空間領域分割を用いた並列処理（プロセス並列）を行う。また、プログラミング言語にはこれまで開発されてきた多くのFortran77のコードからの移行を容易にするため、また(2)(4)の方針のためにFortran90を用いている。



(a) 連続性のある格子接続



(b) 重ね合わせによる格子接続

図1．マルチブロック構造格子

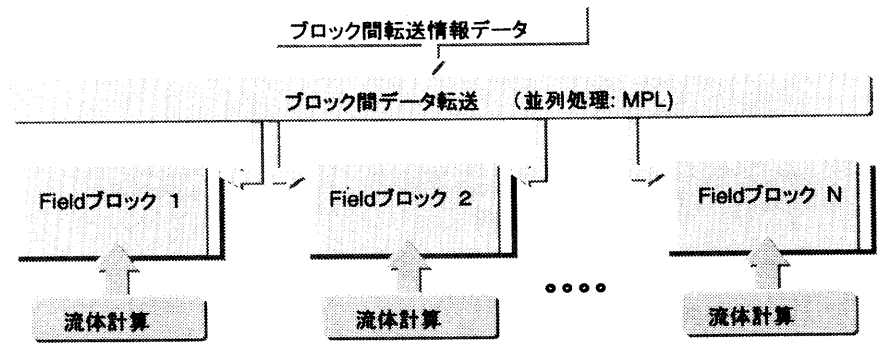
3．マルチブロック構造格子法と並列計算法

マルチブロック構造格子法は、図1(a)のように、完全に連続な格子ブロックの接続によるもの、図1(b)のように重ね合わせを行うものなど、空間をブロック分割する方法すべてを対象にしようとしている。現在、図1(a)のケースについて開発を進めているが、複雑形状への対応という意味では、ブロックの接続を非構造的にすることにより、かなり柔軟に物体形状に適応することができる。ブロック数は数十から数千程度のオーダーまで念頭においている。

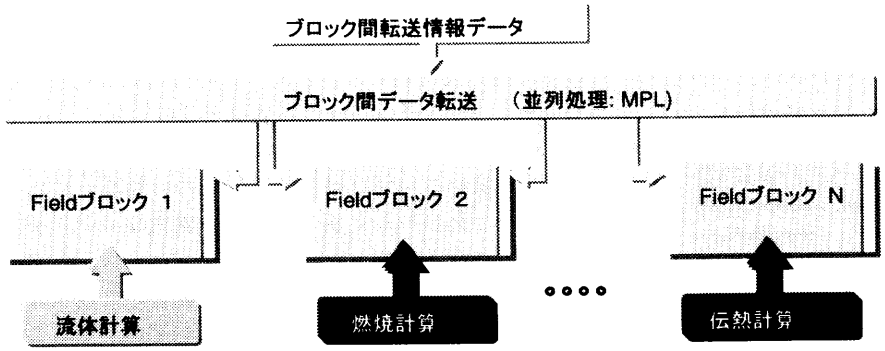
並列計算には領域分割法をもちいる。文献[3][4]と同様に、1つ1つのブロック内部の流体計算には並列処理を持ちこまないようにし、ブロックどうしのデータ交換だけにMPIを利用する。1つのプロセッサが1つ以上のブロック内部の計算すべてを面倒みるようにすれば、ブロック内部の流体計算は従来の単純なシングルブロックの計算となる。ブロック内部の計算が1ステップ終了した段階で、ブロックどうしで必要な物理データの受け渡しを行い、次のステップに計算を進める。

このデータの受け渡しを汎用的に行うには、格子ブロックの接続情報が必要となる。マルチブロックに対応した市販格子生成ソフトウェアによっては接続情報を出力するものもあるが、格子生成コードに依存させないために、格子接続を解析する前処理プログラムを用いている。

格子の接続情報があれば、マルチブロックに対応する流体計算は図2(a)のように、1つのブロックデータに流体計算を行う部分と、ブロック間のデータ転送を受け持つ部分に分離する事ができる。さらに、図2(b)のようにブロックによってソルバーを切りかえることにより、連成計算も可能となってくる。一般にマルチブロック法ではブロックの数と並列計算機のプロセッサの数は一致しないため、ブロック間のデータ転送には、プロセッサ間のデータ通信と、同一プロセッサが対応するデータ転送の2種類を準備する必要がある（図3）。なお、非構造格子法を用いた場合は、1プ



(a) 領域分割による並列化



(b) 連成計算の場合

図2．UPACSの並列計算の概念

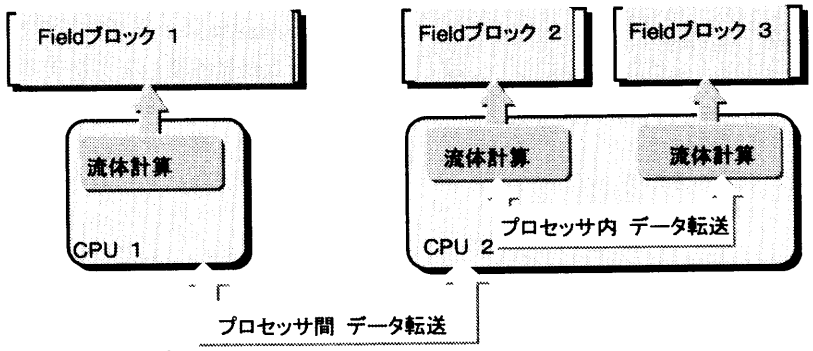


図3．プロセッサへのブロックの割当

ロセッサあたりに1ブロックとなるため、データ転送は1種類でより単純なものになる。

4．コードの基本設計

ブロック内部の計算とブロック間のデータ転送は分離できるが、データ転送のレベルではプロセッサとマルチブロックを意識する必要がある。したがって、データの階層構造に合わせると、ブロック内部の計算を行うシングルブロックのレベルは、マルチブロックのレベルの下層に来る必要がある。そこでUPACSでは、大きく分けて次のような3階層のプログラム構造を取っている（図4）。

(1)最下層：シングルブロック・レベル

ブロック単位の流体計算は、図5のように定義された構造体変数を引数にしたサブルーチンによって、完全にシングルブロックのイメージで計算を行う。シングルブロックについてのCFDのアルゴリズムはほぼ標準化されており、従来のものを移行させることは容易であるし、また、新規に作成したとしてもプログラミング自体は難しいものではない。

このレベルでデータ転送に関連したものとして、そのブロックが関わる接続情報に基づき、転送データをマルチブロック・レベルのサブルーチンとの間で授受する作業がある。これは接続情報に従って処理するだけなので、普通このレベルにおいては、転送すべき物理データの配列（へのポインタ）の指定だけを意識すれば十分である。

また、ブロックの変数全体を1つの構造体にしておくことで、中間層になるマルチブロック・レベルはブロック内部の変数に変更があっても影響を受けないようになる。また、乱流モデルなどの物理モデルが追加されて新たに変数が必要になったとしても、他のプログラムには影響を与えない。

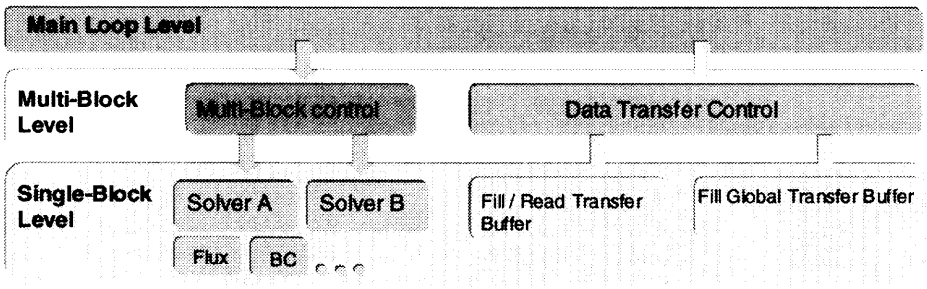


図4．プラットフォームの概念的な階層構造

```
type blockDataType
  use bcTypeModule
  use transferTypeModule
  ..................
  integer:: blockID
                              ! 境界条件情報
  type(bcType),dimension(:),pointer:: boundaryConditions
                                ! データ転送情報
  type(transferType),dimension(:),pointer:: transferSends  &
                                  , transferRecvs
  ..................
  integer :: imax,jmax,kmax   ! データ サイズ
  ..................
  real(8), pointer, dimension(:,:,:,:):: grid   ! 計算格子データ
  ..................
  integer :: nPhys
  real(8), pointer, dimension(:,:,:,:):: q, dq   ! 物理データ
  ..................
  real(8),pointer, dimension(:,:,:)   :: vol   ! メトリック データ
  ..................
end type blockDataType
```

図5．ブロック内データを格納する構造体の例

(2)中間層：マルチブロック・レベル：

ブロック間データ転送、プロセッサへのブロックの割り付けのような、マルチブロック処理を行う。このレベルでは、ブロック内部のデータの詳細に立ち入る必要はないので汎用化することができ、流体計算ソルバーの改良や拡張を行う場合に、何ら影響を受けず、共通に利用する事が可能になる。

(3)最上層：メインループ・レベル：

最上層はコードの骨組みをつくるメインプログラムのレベルであり、図6のようにCFDコードに共通な繰り返し計算のループに並列計算に関わる処理を加えたものになる。このレベルもある程度の一般化は可能と考えられるが、初期化やソルバーによって、ブロック間転送の種類や回数が変わるなど、骨格の設計が変わる可能性があるため、こちらはソルバーを改良するユーザーが見ることを想定して設計する必要がある。

5．プログラミングの概要

実際のFortran90によるプログラミングを考えると、ソルバーを改良または拡張しようとするユーザーには、中間層は一種のライブラリやユーティリティのような形にした方が汎用的に利用し易くなる。そこで、中間層のサブルーチンにはシングルブロック・レベルのサブルーチン名（ポインタ）を引数として渡して、これを実行させるようにした。図7はその概念的なプログラムを示したものであり、例えば、doAllBlocks(step) は、singleBlockSolver という最下層 module にある step(block)というサブルーチンを、各プロセッサが管理している全てのブロックについて実行する、というものである。ここで、block は図5のような構造体変数である。

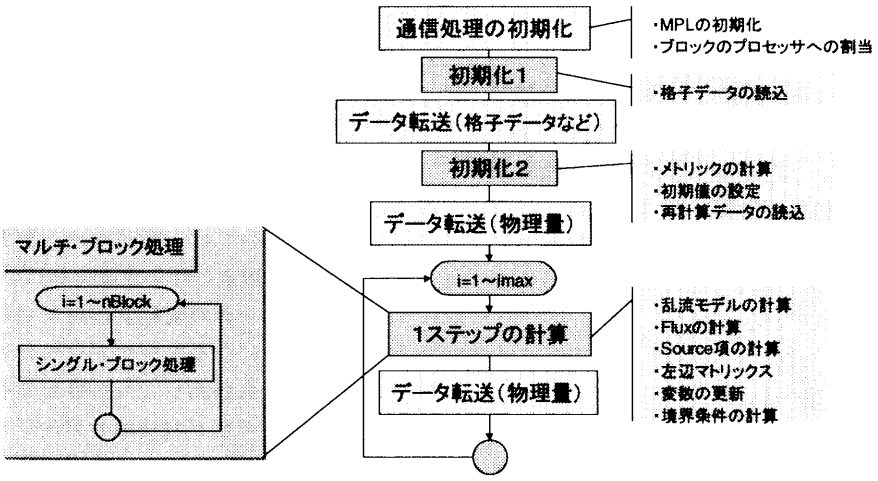


図6．メインループ・レベルのフローチャート

6．パラメータ・データベース

特定の分野に限らずソースレベルでのプログラムの共有化を目指そうとしたとき、計算に必要な計算制御パラメータ、物理パラメータの管理は重要な課題になる。分野によって必要なパラメータが異なってくるために、パラメータを一般化することが難しくなってくるためである。そこで、1つの解決策として、UPACSではデータ管理を専門に行うデータベースモジュールを利用している。パラメータは入力時にすべて「タグ」になるキーワードとともにリストとして管理され、プログラムのレベルに限らず、データベース・モジュールの引用（Fortran90の *use*）により、どこからでも「タグ」を与える事により利用できるようにした。この結果、UPACSのほとんどの部分を利用しながら、異なる計算方法の導入にも柔軟に対応できるようになった。

7．現状と今後の予定

本プロジェクトは昨年度から3年間の計画で進め、以上のようなアイデア、設計手法を積み重ねながら、プロトタイプとなるコードの開発を進めてきており、現在は簡単な問題によるソルバーと並列処理の検証を行っている。今後、ターボ機械流れ、燃焼流や伝熱との連成問題などへの拡張を行い、また並列計算のロードバランシングのための前処理ツールなどを作成しながら、航技研の次世代超音速機技術プロジェクトや通産省の環境適合型次世代超音速推進システム技術（ESPR）プロジェクト等で本格的に利用して行く予定である。さらに将来は、共同研究などを通じて広くコードを利用してもらえるように進めていこうと考えている。

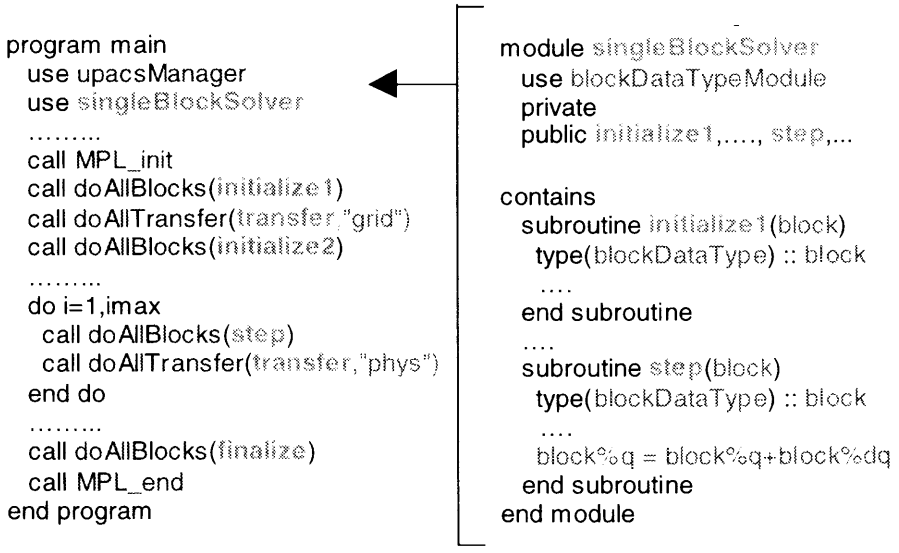


図7．プログラムのイメージ

参考文献

1. "The Message Passing Interface (MPI) standard", http://www-unix.mcs.anl.gov/mpi/index.html.
2. "PVM (Parallel Virtual Machine)", http://www.epm.ornl.gov/pvm/
3. Engel, K., et. al., "Numerical Investigation of the Rotor-Stator Interaction in a Transonic Compressor Stage," AIAA Paper 94-2834, June 1994
4. 太田高志、「並列流体計算のためのオブジェクト指向フレームワーク」、航空宇宙数値シミュレーション技術シンポジウム'98論文集、航技研資料SP-41,1999年2月