CFD解析ソルバーADCSの並列化について

○雷 忠(宇宙航空研究開発機構)，永田 靖典(株式会社 菱友システムズ)

Parallelization of CFD Solver ADCS

Zhong Lei（Japan Aerospace Exploration Agency）and Yasunori Nagata（Ryoyu Systems Co.,Ltd.）

Key Words : CFD, Parallel Programming, Compressible Flow

Abstract

A multi-block CFD solver developed by JAXA was modified for parallel computation. Parallel programming MPI library was utilized to improve the performance and the applicability for large-scale problems. The parallelized code was verified and its performance was tested. It shows that the code is efficient to conduct large-scale computations by using MPI parallelization.

1. はじめに

流体工学、流体機械の研究開発において、数値解析の大規模化や形状の複雑化が進む一方、より短期間、高精度で解析を行うことが要求されている。大規模な計算を行うためには、計算時間の制約からコードの並列化が必要不可欠である。一方、科学技術計算を支えるハードウェアもますます大規模な並列計算機システムとなっている。ハードウェアの中核を成すCPUはマルチコア化が進み、省電力化を目的として今後もその方向性が変わることはないと思われる。並列計算機システムにおいては、並列処理をいかに効率良く行うかが重要であり、システムのリソースを無駄に占有することは避けなければならない。

航空機の空力設計では、従来、風洞試験の繰り返しと線形理論設計法の併用により行われてきたが、CFD（Computational Fluid Dynamics）は革新技術として不可欠であり、今後、設計に向けて大きな役割が期待されている。新しい航空機の開発を成功させるために、如何に開発期間を短縮しコストを低減するかは重要となる。しかし、空力解析においては非線形偏微分方程式である流れの支配方程式が複雑であり、特に予測精度が要求される空力設計では計算負荷が依然高い。そのため、計算機性能の向上とともに、解析ソフトウェアの高速化が必要不可欠である。

JAXA超音速機チームでは、独自にCFD解析ソルバーADCSを開発し、これまで、低速から高速まで様々な流れの解析を行った。ADCSは有限差分法により流れの支配方程式を離散化し、複雑な形状に対応するマルチ・ブロック技術を採用した[1]。このコードはすでに並列化が施されていたが、コード設計段階での制約から適用限界が存在し、並列計算機システムの性能を十分に発揮出来ずにいた。そこで今回、マルチ・ブロック構造格子をあらかじめ分割して、それぞれの領域を割り当てられた各CPUで処理する領域分割法に基づいた並列化を施し、領域間の情報交換にはMPI（Message Passing Interface)ライブラリを用い、Fortran90の構造体機能を用いることにより、大規模計算に対応させるとともに、計算の高速化とコードの汎用化を図った。

2. ADCS概要

ADCS（Aero-Dynamic Computational System）は、JAXA超音速機チームで開発したCFD解析ソルバーである。

流体は圧縮性完全気体を対象とし、流れ場の支配方程式は3次元レイノルズ平均ナビエ・ストークス（RANS）方程式とした。乱流解析には渦粘性モデルであるSpalart-Allmaras（SA）モデル[2]、Menter's SST k-ωモデル[3]と数種類のk-εモデルを組み込んだ。空間離散化には有限差分法を用いている。対流項の評価にはChakravarthy-Osherスキーム[4]を用いて風上化し、3次精度のMUSCL補間法により高次精度化を施した。粘性項の評価には2次精度中心差分を用いた。時間積分には対流項と粘性項を含めたLU-ADI近似陰解法を用い、収束を加速するために、局所時間刻み法を適用した。複雑形状にも対応できるようにマルチ・ブロック構造格子を採用しており、3次元、2次元流れについて計算が可能である。

ADCS内は図 1に示すように前処理とソルバー、後処理で構成されている。前処理の段階で各ブロックの仮想格子点を作成し、非計算点を抽出し、乱流解析に必要な最短壁距離を計算して、これらの情報は一旦ファイルに出力される。ソルバーは前処理で出力されたファイルを読み込み、離散化された流れ場の支配方程式に対する解析を行う。改良後のADCSでは、前処理の段階で並列計算のためのCPUの割当を行い、後処理の段階でソルバーからCPU毎に出力された複数の結果ファイルの結合を行う。

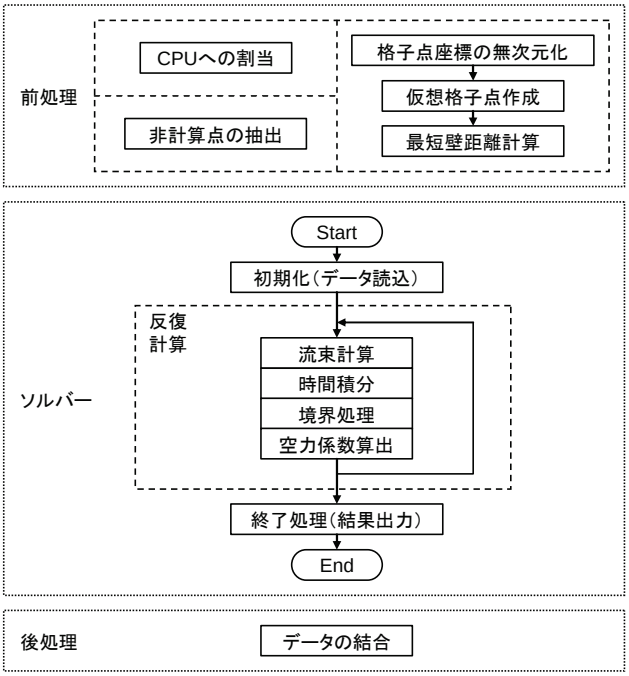

図 1 処理内容と処理の流れ

3. コード改修内容

3.1. 並列化

並列化の方法としては、一連のタスクを複数のCPUに割振る機能分割法と、データを複数のCPUに割振り、個々のデータに対して処理を行う領域分割法の考え方がある。大規模CFD解析の場合は、データ量が膨大であるため、計算領域を予め分割し、それぞれの領域を割り当てられた各CPUで処理する領域分割法に基づいた並列化がよく用いられる。

改良前のADCSでは、各ブロックについてある特定の方向を分割することで並列処理を行っていた。この場合、この方向での流束評価を行うたびにCPU間の通信が必要となり、1回の反復計算で何度も通信を行わなければならない。また、ブロックによって領域分割方向の格子点数が大きく異なる格子の場合には、一部のCPUに処理が集中し、他のCPUは待ち状態の割合が大きくなってしまう。これではリソースを有効に活用することができないので、格子生成時に各ブロックの領域分割方向で格子点数のばらつきに注意しなければならなかった。

ADCSはマルチ・ブロック構造格子に対して計算を行うため、改良後のADCSでは各ブロックをCPUに割り当てることで領域分割し、並列処理することとした。1つのブロックに対する処理を1つのCPUに割り当てることで、流束計算、時間積分の処理は1つのCPU内で閉じることになる。そのため、反復計算時のCPU間の通信は境界処理、空力係数算出でのみ行われることになり、通信量を削減できる。通信回数と通信量は計算速度に関係するため、これらを削減することで計算時間が短縮される。また、ブロックのトポロジーと計算効率の関連がなくなるため、格子生成時にブロックのトポロジーを考慮する必要がなくなり、それぞれのCPUに割り当てられる格子点数を揃えることで、計算負荷を均等にできる。

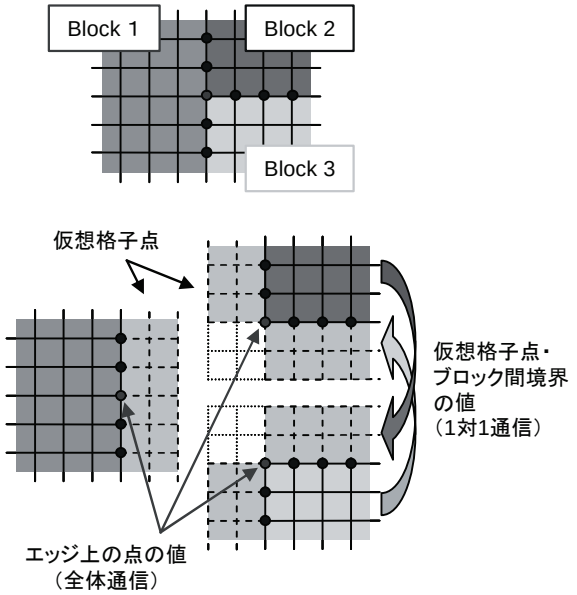

図 2 境界処理時のCPU間通信

改良後のADCSではMPI（Message-Passing Interface）ライブラリを用いて並列化を行った。MPIは分散メモリ型並列処理のためのライブラリの規格である。MPIは現在、ほとんどの並列システムに実装されており、規格が明確に定義されているため、非常に汎用性が高いものとなっている。また、通信を行うタイミングを明確に指定できるなど、細かい制御が可能である。このため、MPIライブラリを用いて並列化を施し、CPU間の通信処理を最適化することで、コードの汎用性を高めるとともに計算の高速化を図った。

改良後のADCSにおいて、空力係数算出では全体通信、境界処理では1対1通信と全体通信を用いて、CPU間のデータのやり取りを行っている。MPIでは、全てのCPUに対して通信を行う全体通信と個々のCPU間で通信を行う1対1通信がライブラリに含まれる命令によって実行される。空力係数の算出では機体表面について積分を行うため、全てのCPUの計算結果を用いる必要がある。一方、境界処理では図2に示すように隣接するブロックの物理量を仮想格子点の値として設定するため、通信が必要なのは隣接するブロックがそれぞれ割り当てられたCPU間である。また、ADCSは物理量を節点に置く、節点ベースのソルバーであるので、境界面上の値についてもブロック間で一致させる必要がある。境界面の角にある節点については、複数のブロック間で値を一致させる必要があるため、全体通信を別に行っている。

3.2. 省メモリ化

メモリもCPUと並ぶ重要な計算機システムリソースであり、効率良く運用することが求められる。

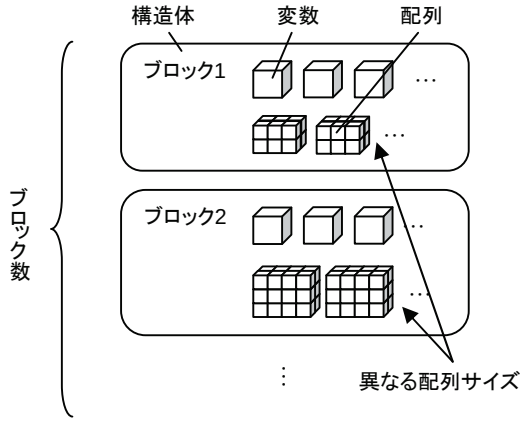

図 3　構造体を用いたデータ構造

旧ADCSでは使用されないメモリ領域を確保してしまう問題があり、省メモリ化が求められていた。これまでは、各格子点のデータは4次元配列（3方向+ブロック番号）で管理し、全てのブロックについて同じ点数分の領域を確保していた。しかし、マルチ・ブロック構造格子は一般に各ブロックの形状が異なっているため、各方向について格子点数のばらつきがあると、メモリを余分に確保していた。また、解析対象によって格子点数は異なり、解析対象に合わせて配列の大きさを変えるには、コードを手動で修正する必要があるため、手間がかかっていた。メモリを余分に確保すると、配列の参照に時間がかかり、計算性能の低下につながる。

そのため、プログラム言語をFortran77からFortran90へ移行し、Fortran90で追加された機能である配列の動的割り付け、および構造体を用いることで省メモリ化を図った。改良後のADCSでは各ブロックのデータを一つの構造体に集約し、構造体内に3次元配列を持たせたデータ構造（図 3）を採用した。構造体は動的に確保され、構造体内の3次元配列もそれぞれ動的に確保される。構造体内の3次元配列は構造体毎に別々の大きさを設定できるため、メモリを過不足なく確保できる。ブロック数や確保すべき配列の大きさは、プログラム実行時に格子データから読み取るため、配列の大きさを変えるためにコードを修正する必要はない。このように個々のブロックについて動的に確保することで、余分なメモリ確保を無くし、コード修正を行わなくても様々な格子に対応できるようになった。

3.3. 高速化

コード改修に伴い、以下の項目について検討し、コードの最適化と高速化を試みた。

- CPU間通信に要する時間の削減
- 演算量の削減
- 演算速度の向上
- 入出力の高速化

CPU間の通信には、通信を行うための処理時間と通信

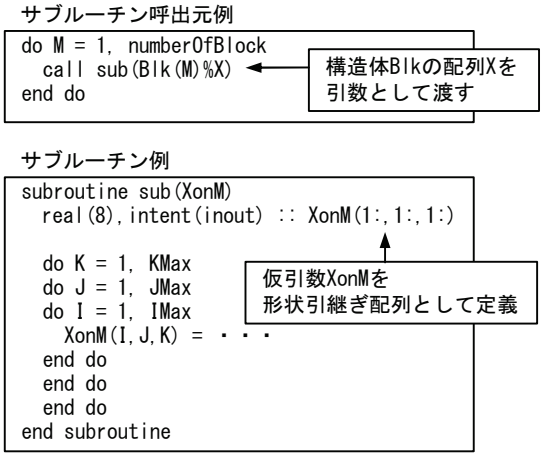

図 4　配列参照の高速化

完了までの待ち時間が発生する。前者は、データをまとめて1回の命令で送り、通信に関する命令の回数を減らすことで削減できる。後者については、通信回数と通信データ量を削減することで削減できる。計算時間を短縮するためには、演算量そのものを削減するとともに、演算速度が速くなるようなコーディングが必要となる。旧コードでは同様の処理を重複して実行している箇所があったため、重複を無くして1回の処理で済むようにした。また、計算負荷の大きい処理について無駄な計算を無くすなど、最適化を施し、演算量の削減を図った。演算速度にはメモリ参照時のキャッシュ・ミスの発生率と、配列の参照に要する時間が大きく関係する。これらはコーディングを工夫することで対処した。配列参照速度については以下で述べる。一般的に入出力の処理は遅い処理である。そこで、並列処理の利点を生かして、入出力においても各プロセスが別々のファイルに対して同時に入出力するようにした。これは多数のストレージ・デバイスを有する並列計算機システムにおいては有効であると考えられるが、単一のストレージ・デバイスしか持っていないWSやPCでは大きな効果は期待できないと思われる。

一般に、大きさが固定された配列を動的に確保された配列に置き換えると、計算が遅くなる傾向があるなど、データ構造は処理速度に影響を与える。今回のADCSの改修においても、データ構造を変更したことで計算速度が大幅に遅くなる問題が生じた。

この問題に対処するために、演算ルーチンを呼び出す際に必要なデータを引数として与え、ルーチン内では形状引継ぎ配列としてデータを参照する方法（図 4）を採用した。形状引継ぎ配列を用いることに関連して、全てのルーチンをモジュール化し、デバッグにかかる労力も軽減させた。この方法を用いることで構造体を参照する段階が省略された結果、構造体をそのまま用いたコードに比べて計算時間が約35%短縮され、演算速度が改善した。

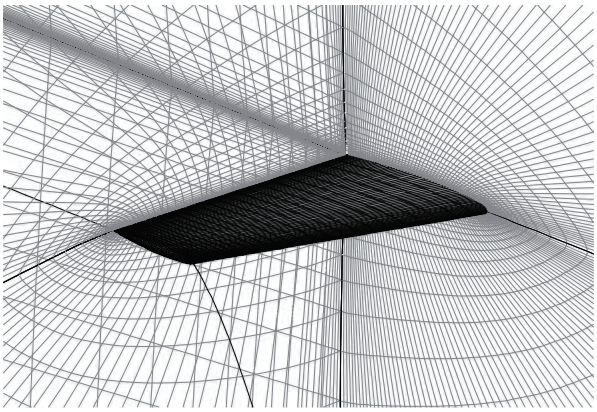

図 5 ONERA M6 翼の計算格子

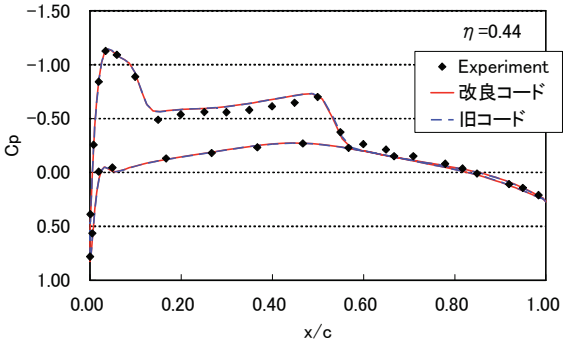

図 6 翼断面Cp分布（η=0.44）

4. 並列化・改修の検証

旧コードとの比較のため、ONERA（フランス航空宇宙研究所）の3次元遷音速翼M6まわり流れの数値解析を行った。また、並列計算による解析結果への影響の有無についても確認を行った。

4.1. 解析条件

ONERAのM6翼に対して遷音速条件で流れの解析を行った。図 5に使用した計算格子を示す。格子点数は約39万点、分割ブロック数は4である。マッハ数0.8395、迎角3.06[deg.]、レイノルズ数Re=11.72×10^6の風洞試験に合わせて、SA乱流モデルを用いた全面乱流の数値計算を行った。

4.2. 計算結果

図 6に、改良コード、旧コードそれぞれを用いて計算した半スパン長の44%位置での翼断面圧力係数Cp分布、およびその実験値を示す。改良後のコードは旧コードと結果が完全に一致しており、実験値にも近いことがわかる。さらに、改良コードについてプロセス数を1,2,4と変えて逐次・並列計算をそれぞれ行ったところ、結果は全て一致した。よって、改良コードは旧コードと同じ結果を示し、並列化による計算精度への影響は無いことが確かめられた。

5. 並列性能の新旧比較

超音速機まわりの計算を旧コード、改良コード両方で行い、並列性能の改修による効果を調べた。

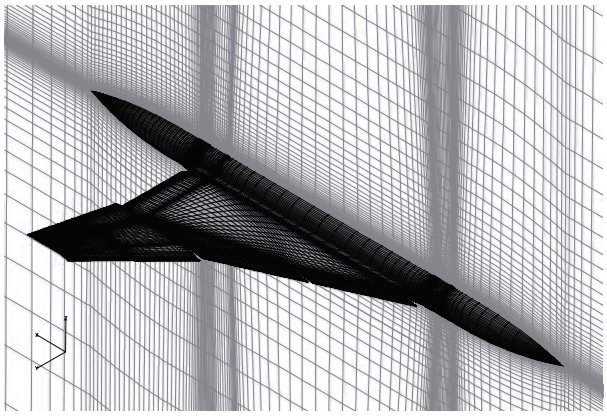

図 7 JAXAジェット01次形状の計算格子

5.1. 解析条件

JAXAジェット実験機01次形状高揚力形態に対して亜音速条件で流れの解析を行った。図 7に数値解析に使用したマルチ・ブロック計算格子を示す。総格子点数は約430万点、分割ブロック数は計算効率を考慮して、旧コードに対しては8、改良コードに対しては72とした。風洞試験の条件に合わせてマッハ数0.088、迎角12.0[deg.]、レイノルズ数Re=0.945×10^6で計算を行った。この流れ解析には乱流モデルとしてMenter's SSTモデルを用いた全域乱流を仮定した。

5.2. 並列計算性能測定環境

計算はJAXAスーパーコンピュータシステム（JAXA Supercomputer System, JSS）で実施された[6]。現在運用されているJSS第1期導入システムのハードウェアの主な諸元を表 1に示す。このシステムでは、1コアに対して1プロセスを割り当てるFLAT並列と、1CPUに対して1プロセスを割り当て、1プロセスを4コアでスレッド並列処理するIMPACT（Integrated Multicore Parallel ArChiTecture）並列が使用できる。IMPACT並列のスレッド並列化は、コンパイラの自動並列によってコードを書き換えることなく使用できる。ここでは、両コードともFLAT並列について並列性能の計測を行った。

5.3. 並列処理性能

使用するCPUのコア数を変えて、図 1に示された反復計算100回に要する時間およびそのときのメモリ使用

表 1 測定環境の諸元

ハードウェア	Fujitsu SPARC Enterprise M9000
演算性能	1.28 TFLOPS、40 GFLOPS/CPU
メモリ容量	1 TBytes
CPU数	32（128コア）
アーキテクチャ	SMP
CPU	SPARC64 VII
L2キャッシュ容量	6 MBytes/CPU
コア数	4コア/CPU

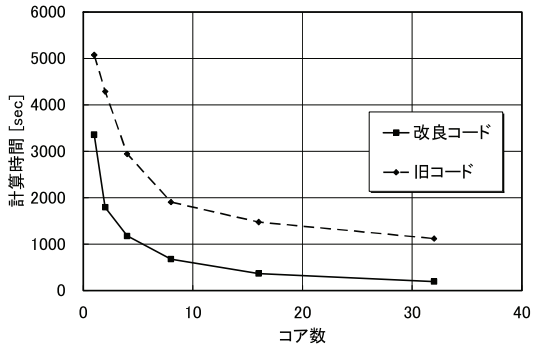

図 8　計算時間の新旧コード間比較

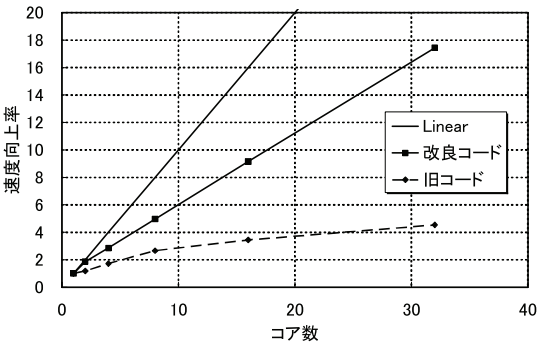

図 9　速度向上率の新旧コード間比較

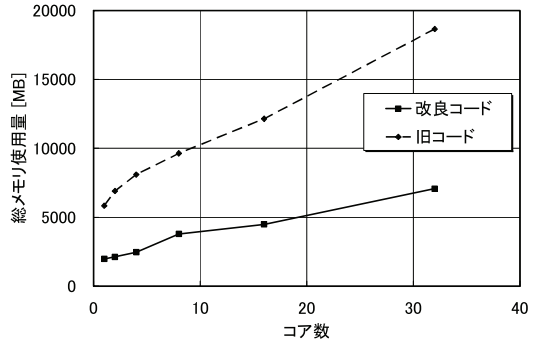

図10　メモリ使用量の新旧コード間比較

量を計測した。使用したコア数に対する計算時間を図8に、並列化による速度向上率を図 9に、総メモリ使用量を図10にそれぞれ示す。並列化による速度向上率は次式で求めた。

$$(\text{速度向上率}) = \frac{(\text{計算時間})}{(1\text{コア使用時の計算時間})} \qquad (1)$$

両コードとも使用するコア数を増やすと、計算時間が短くなっていることがわかる。また、全体的に改良コードの方が計算時間が短く、コア数が1の並列処理を行わない場合で約34%、コア数が32の場合では約83%計算時間が短縮された。両コードの速度向上率を比較すると、改良コードの方が速度向上率が大きくなっており、並列処理性能が大幅に改善されていることがわかる。総メモリ使用量について比較すると、旧コードに比べ改良コードはメモリ使用量が削減されていることがわかる。このテスト問題の計算では、約65%削減されていることがわかる。なお、総メモリ使用量がコア数とともに増加するのは、各ブロックの計算で共通に使用する一時配列をコア毎に確保するためである。以上のことから旧コードで問題であった計算効率とメモリ使用効率について大幅な改善が達成されたことが確かめられた。

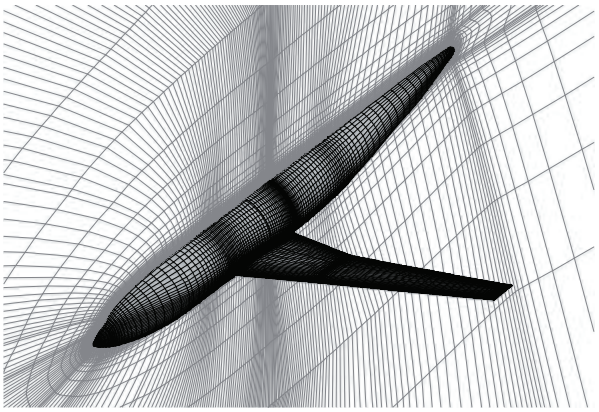

図 11　DPW-3の計算格子

6. 並列処理性能のテスト

改良コードの大規模計算への対応の可否と並列処理性能について検討するために、航空機まわり流れの数値解析を行った。

6.1. 計算条件

解析対象はAIAA（米航空宇宙学会）の3rd Drag Prediction Workshop[5]で用いられた亜音速旅客機の翼胴形態である。図 11に使用したマルチ・ブロック計算格子を示す。総格子点数は約980万点、分割ブロック数は222である。マッハ数0.75、迎角1.0[deg.]、レイノルズ数Re=5×10^6で計算を行った。乱流モデルとしてSAモデルを用い、全域乱流を仮定した。

6.2. 並列計算性能測定環境

計算は前節で述べたJSSで実施された。ここでは、FLAT並列、IMPACT並列両方について並列性能を計測し、両者の比較も行った。

6.3. 計算結果と並列処理性能

図 12に機体表面圧力係数Cp分布を、図 13に機体中心面から半スパン長41.1%位置での翼断面Cp分布をそれぞれ示す。これより、翼面上で全幅に渡って、衝撃波が発生しており、横41%位置では前縁から30%局所コード長位置付近で発生していることがわかる。これらの結果は、3rd Drag Prediction Workshop[5]に参加した他の計算結果とよく一致していることが確認されている。コードの信頼性を確認するためにはさらに多くの検証計算を行っていく必要があるが、本コードが大規模計算に対応できることが確認できた。

同じ格子と流れ条件を用いて、使用するCPUのコア数

図 12 機体表面Cp分布

を変えて50回の反復計算に要する時間を計測した。図14に使用したコア数に対する並列化による速度向上率を示す。使用するコア数を増やすと、速度向上率が増加し、計算時間が短くなっていることがわかる。FLAT並列とIMPACT並列とを比較すると、同じコア数を使用する場合、FLAT並列の方が計算が速いが、大きな差はないことがわかる。

図 14には参考としてアムダールの法則から求められる速度向上率も示している。アムダールの法則は並列計算の理論上の性能予測によく用いられており、速度向上率は次のように求められる。

$$(\text{速度向上率}) = \frac{1}{(1-r)+r/p} \qquad (2)$$

ここで、pはプロセス数であり、コア数に対応する。rはコードの並列処理部の割合である。本コードについて並列処理部の割合を見積もった訳ではないが、計測結果に近い値を通るr=0.985としたときの速度向上率を図 14に示す。アムダールの法則には通信による遅延や待ち時間が含まれておらず、計測結果と単純に比較することはできないが、本計算の場合、並列処理部の割合が98.5%に相当する高い並列性能を本コードは達成しているといえる。

7. まとめ

JAXA超音速機チームで開発したCFD解析ソルバーADCSに対して、領域分割をブロック単位に、並列化手法をMPIにそれぞれ変更し、プログラム言語をFortran77からFortran90へ移行して、大規模計算への対応と計算の高速化、およびコードの汎用化を施した。並列化による計算精度への影響がないことを確認した。新旧両コードに対して同じ計算を実施し、旧コードに比べ計算時間、メモリ使用量が少ないことを示した。航空機まわりのCFD計算を実施し、大規模計算が可能であることを確認した。使用するCPUのコア数を変えて速度向上率を計測し、高い並列処理性能を有していることを示した。

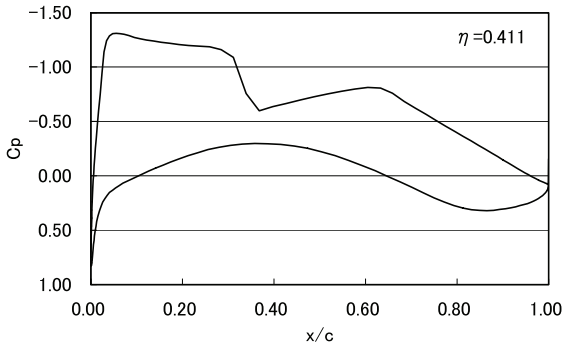

図 13 翼断面Cp分布（η=0.411）

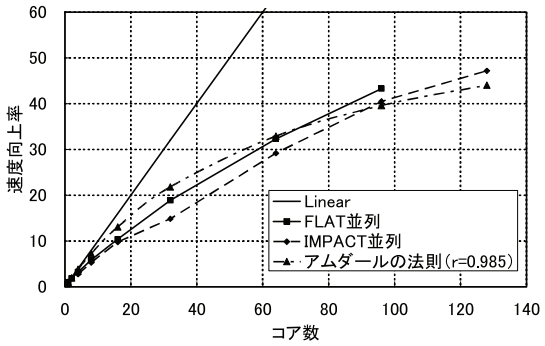

図 14 速度向上度の計測結果

た。

今後さらに検証計算を進め、コードの信頼性を高めていく予定である。

謝辞

本研究を行うにあたり、宇宙航空研究開発機構情報・計算工学センターの計算機を使用させて頂きました。

参考文献

1. 雷忠, “超音速機高揚力装置に関する数値解析及び考察”, 宇宙航空研究開発機構報告, JAXA-RR-07-050.
2. Spalart, P.R. and Allmaras, S.R., “A One-Equation Turbulence Model Aerodynamic Flows,” AIAA paper 92-0439, 1994.
3. Menter, F.R., “Zonal Two Equation k-ω Turbulence Models for Aerodynamic Flows,” AIAA paper 93-2906, 1993.
4. Chakravarthy, S.R. and Osher, S., “A New Class of High Accuracy TVD Schemes for Hyperbolic Conservation Laws”, AIAA paper 85-0363, 1985.
5. 3rd AIAA CFD Drag Prediction Workshop, June 3-4, 2006, http://aaac.larc.nasa.gov/tsab/cfdlarc/aiaa-dpw/.
6. JAXAスーパーコンピュータシステムについて, 2008年3月28日, https://www.jss.jaxa.jp/JX/manual/forum_vol1.pdf.