JAXAデジタル/アナログ・ハイブリッド風洞システムに関する予備技術検討

佐藤幸男[1], 松原聖[1], 渡辺重哉[2]
1アドバンスソフト株式会社, 2宇宙航空研究開発機構

Preliminary Study on the System Architecture of JAXA Digital/Analog Hybrid Wind Tunnel

by
Yukio Sato and Kiyoshi Matsubara (Advancesoft Corporation)
Shigeya Watanabe (Japan Aerospace Exploration Agency)

ABSTRACT

JAXA has a plan to develop the system which integrates EFD and CFD. This system has a database which stores both EFD data and CFD data. The user can obtain advanced information by searching, visualizing, and comparing EFD data and CFD data stored in the system. We report the system configuration, issue, and data format. Although the details of data format will be decided in the near future, we report an example about containing data in a data format.

1. 緒言

JAXAでは、風洞実験とCFDを融合したシステム（デジタル/アナログ・ハイブリッド風洞）の構築を計画している。そのシステム実現のための技術検討のうち、本報告では、システム構成例とその課題、および、本システムの基盤となるデータフォーマットの2項目について報告する。

2. システム概要

JAXAハイブリッド風洞システムは初期段階として完成後10年程度の運用を想定しており、新規に登録されるデータは毎年10テラバイトオーダーの量になると予想され、大容量のデータをハンドリングできるデータベースシステムが必須である。本システムの機能は、実験データやCFDの結果等をユーザがデータ検索・可視化・比較を行うことで、膨大なデータから抽出した高度な情報を提供するとともに、両データを統合的にデータベースに蓄積することである。

本システムでは実験データとCFDデータの一元管理とユーザの利便性の両立を図るために「ゆるやかに」データの標準化を行うことが求められる。また新規の計測手法などへ対応しうる拡張性が必要である。

図1に本システムの概念図の一例を示す。ユーザは、本システムをユーザのPCから利用でき、将来的にデータ等をWEBで公開するシステムを想定している。本システムの基盤部分は、(1)ファイルモジュール、(2)データベースモジュール、(3)可視化モジュール、(4)WEBモジュールの4つのモジュールから構成される。

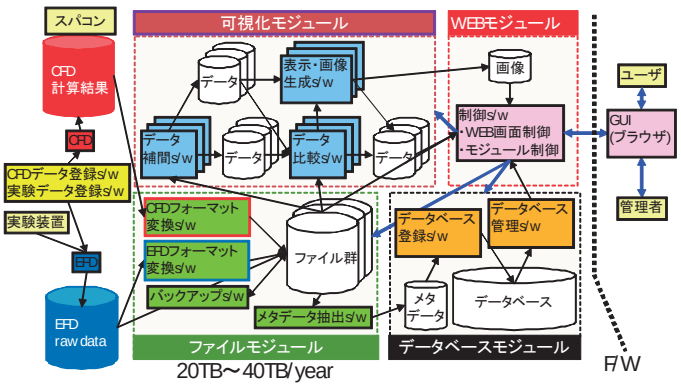


図1 ハイブリッド風洞構成例

(1)ファイルモジュール

本モジュールでは、ユーザが画像や数値を含む各種の実験データを保存する。図2に示すようにCFDと風洞実験のデータは本モジュール内で集中管理する。後のデータ検索を容易に行えるようにデータそのもの(実データ)だけではなく、検索に使うキーワードなど(メタデータ)も格納する。メタデータの保存をする際に独立のファイルとして保存する「分離型」と対応するデータと同一のファイルに格納する「非分離型」が考えられる。分離型の場合には実験機器や計算機が出力したファイルを無加工なまま保存するため、その分処理速度が高速になる一方、1つの解析対象に関するデータが複数のファイルに分かれてしまうため、メタデータを記述したファイルが何らかの理由で失われた場合に実データがどのような条件で行われた実験のデータなのか、といった情報が失われてしまう恐れがある。非分離型では1つのファイルに全てのデータとメタデータを格納するため、ファイルそのものが失われない限り実験条件や計算パラメータ等の情報だけが無くなってデータの識別が困難になると言ったことは起きない。一方何らかのファイルフォーマットを用いて全てのデータを保存することとなるため、変換処理を行う必要があり、その分処理速度の低下が発生する。

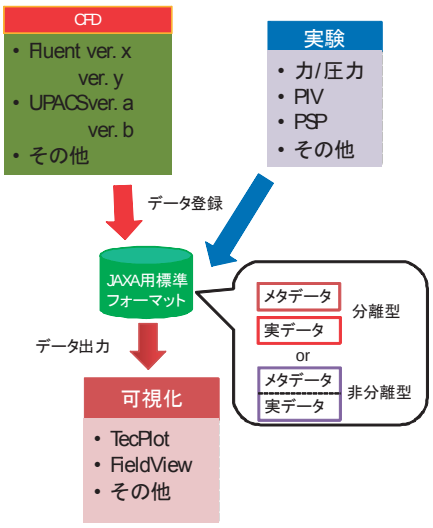


図2 CFD・EFDデータ格納用フォーマット

「分離型」、「非分離型」共に一長一短ではあるが、実験データの補正処理などにミスが発生した場合にもやり直しがきくという観点から実験データについては分離型が有利であると言える。逆に CFD データについては再計算のコストが高くない限りは非分離型が有効であると言える。

(2)データベースモジュール

このモジュールでは検索のキーとして利用するデータ部分を保持する。ファイルモジュールが生成したメタデータ部分を格納し、検索に利用する。なお、検索のキーとして利用するキーワード・項目は新規の実験・シミュレーションへの対応できるように拡張可能であることが求められる。

(3)可視化モジュール

このモジュールはファイルモジュールに格納された風洞実験・CFD データの可視化を行う。風洞実験データの各種干渉補正処理や CFD の補間処理を行った上で、風洞実験と CFD の相互比較検証処理を本モジュールが担当する。例えば実験データについては図 3 のように PSP の欠損値の補正や衝撃波の検出を行うためのエッジ検出処理などが例としてあげられる。CFD については補間処理を行って模型上の圧力ポートの位置における物理量を算出するといった処理が行われる(図 4)。

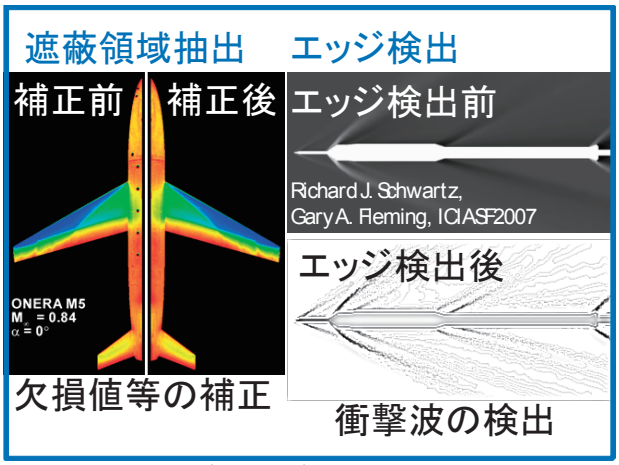


図 3 実験データの補正処理概念図[1],[2]

図 4 CFD データに基づく物理量の補間処理概念図

各種補正処理が完了した後に EFD と CFD を比較する処理が行われる。例としては図 5 のように流れ場の中の渦の位置や形状を EFD と CFD の双方について行うといったことが考えられる。このような処理の実現には特徴点抽出やテンプレートマッチングといった画像処理手法を適用する。これらの手法は物理量の数値データではなく画像データをもとに処理を行うため、実験で得られた撮影データを数値化せずに処理可能であり、CFD のデータについても可視化後の画像に対して適用すればよいため、速度ベクトルの数値がなければ処理出来ない、といったことがない分だけ適用範囲は広く、定性的な比較を行う上では有用である。

定量的な比較を行う場合には補間処理などであらかじめ実験と CFD の数値データを直接比較可能な状態に処理した上で図 6 のように 1 つの図で可視化する。図 6 では機体表面上の pressure tap における圧力係数の値を CFD と実験の双方について比較している。白線がグラフのベースラインで、機体表面の対応するタップの位置と直線で結ばれている。

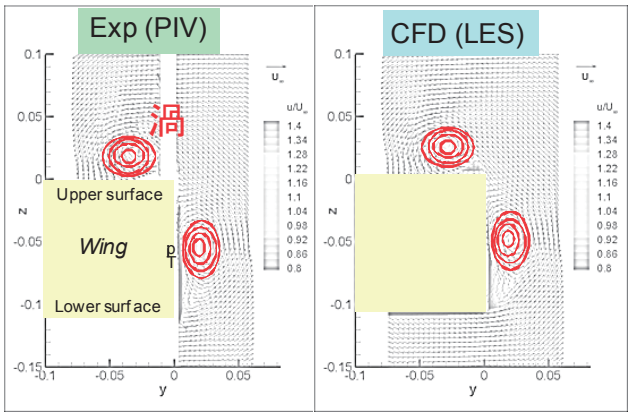


図 5 EFD と CFD のデータ比較概念図[3]

(4)WEB モジュール

本モジュールはユーザの PC から命令を直接受け取り、各モジュールの制御を行う。データの検索、可視化モジュールが生成した画像の表示が行われる。ユーザは図 7 のように WEB ブラウザ上から風洞実験のデータと CFD のデータの比較を行うことが可能である。

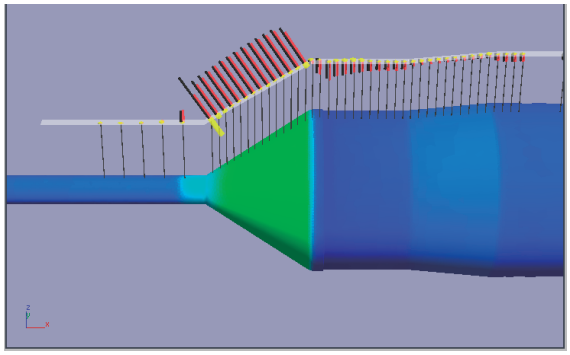


図 6 数値データと実験データの比較例[2]

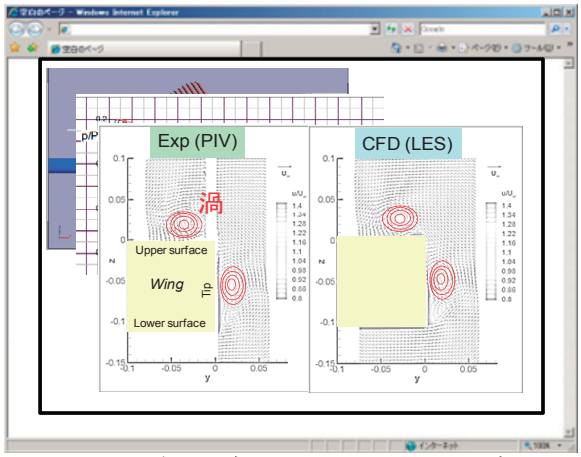


図 7 Web ブラウザ上での EFD と CFD のデータ比較

本システムでは素早く目的のデータが取り出せることが求められているが、データを取り出すことには2つの意味合いがある。1つ目は目的の実験・シミュレーションに関するデータのセットを手早く取り出すことであり、もう1つはあるファイルの中から補間処理などに必要な最低限のデータのみを抽出するということである。1つ目に関してはデータベースモジュールに実験やシミュレーションに関する条件などのパラメータをデータベースモジュールに格納することで対応可能であり、2つ目に関しては先述したJAXA標準フォーマット内部でのデータの格納方法を工夫することで実現する。本報告ではIT技術の側面からどのようにしてデータ検索の高速化を図るか、どのようなフォーマットでデータを蓄積するかの2点について検討した結果を報告する。

3. ファイルフォーマットの検討

先述したように実験条件・実験データとCFD条件・解析結果等のファイルについては、そのデータの特徴を示す部分（メタデータ）と数値データを一体化する方式と分離する方式が考えられるが、ここではメタデータと数値データを分離するシステム構成とする。なおここで分離対象となるデータは実験のrawデータである。いったん補正処理などを施したデータを作成すれば通常rawデータは使用する必要がないが、補正処理に不具合があった、より高精度な補正手法が開発されたので、過去のデータについても適用したい等といった場合にデータ源に立ち返る必要があるため、必ず保存することとする。この目的を満たすにはrawデータを無加工のまま保持する必要があることから、rawデータのみ分離することとする。

メタデータと数値データを分離する方式の特徴としてはデータベースそのものが保持するデータ量を最小限に抑制すること、必要な情報をデータベースとして独立させ、なおかつファイル名を検索のキーとして使用するため、大容量のファイルを格納しても検索の速度は劣化しないということがある。なお、CFDのデータについては入力ファイルがあれば容易に計算結果を再現できることから、実験データのようにCFDコードが出力したファイルそのものを保存する必要性は薄い。したがって変換プログラムを用意してCFDコードの出力したデータをJAXA標準フォーマットへ変換する、あるいはCFDコードにJAXA標準フォーマットによるデータを出力する機能を実装するといった方法が考えられる。

rawデータ以外の実験データとCFDデータ、メタデータを格納するためのデータフォーマットとして有力な候補としてXML[4]、HDF5[5]、CDF[6]があげられる。XMLは文書のデータ構造を定義するための言語で、科学技術計算に限らずあらゆる分野で広く採用されている。HDF5およびCDFについては科学技術計算のデータ格納を想定して開発されたデータフォーマットである。表1にこれらについて比較した結果を示す。これらのフォーマットには後々の新規実験やシミュレーションへ対応しうる拡張性と大容量のデータ入出力に耐えうるIO性能が求められる。どのフォーマットも階層構造によるデータ格納方式を採用しているため、フォーマットの拡張性という点では差が見られない。一方IO性能に関してはHDF5が最も高性能であり、その点においてハイブリッド風洞については適合性が高いといえる。

表1 ファイルフォーマット比較

項目	柔軟性	容量制限の有無	読み書きの速度	採用実績、将来性	総合評価
XML	○	×	×	○	×
HDF5	○	◎	○	◎	◎
CDF	○	○	不明	△	△

4. データベースの検討

データベース(DB)についても現在用いられているリレーショナルDB(RDB)、XMLDB、XML対応RDBの3種について比較した結果を表2に示す。データ格納の柔軟性・拡張性の点においてXMLDBの適合性が高い。XMLDBは速度の点でRDBに劣るが、メタデータのみを抽出することでデータベースの負荷軽減を図ることとする。なお、表2中の項目の傾向はあくまで大まかなものであり、最終的には個別の製品単位での比較・検討次第ではXML対応型のRDBが最適となる場合もあり得る。

表2 データベース比較

種類	柔軟性・拡張性	検索速度	実績・信頼性	コスト	総合評価
RDB	×	◎	◎	◎	△
XMLDB	◎	△	○	×	○
XML対応RDB	○	○	△	×	△

5. データ記述例

前節までの検討内容によりシステム内から目的の実験・CFDに関する一連のデータを取り出すことが可能となるが、さらにその中から個別のデータを効率よく取り出す方法について本節では検討する。

まず実験データとCFDデータを統一的なフォーマット(HDF5)を用いて実際に保存した例について述べる。この結果を踏まえた上で個別のデータを効率よく取り出すことが容易となるデータの格納方法について検討した結果を報告する。HDF5形式のファイル作成方法は以下の2つの方法がある。

(1)HDF専用ビューワプログラム(HDF View)を用いて作成

HDF Viewは基本的にはHDF形式のファイルを閲覧するためのソフトウェアであるが、ファイルの編集機能を有しているため、手動で**図8**に示すようなGUI画面を操作してファイル内に個々のデータを追加する。

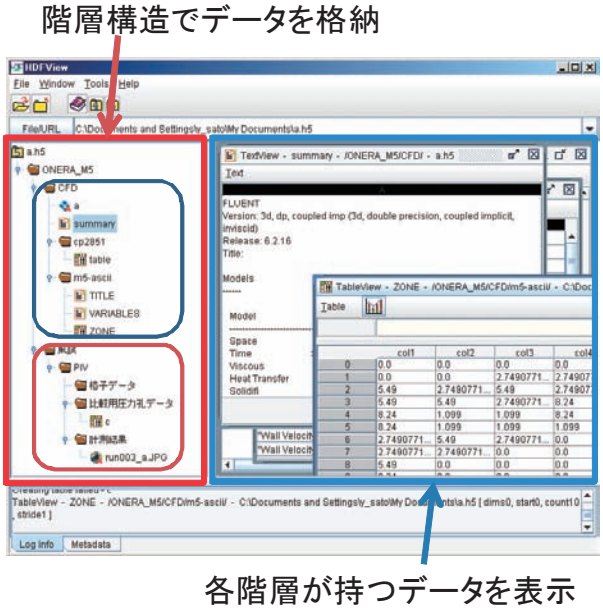


図 8 HDF View[7]による HDF5 形式の閲覧

どのようにデータが格納されているか目視で確認できるため分かりやすいのが HDF View の特徴である。HDF5 は階層構造でデータを格納するファイルフォーマットであり、ユーザが任意の階層構造を設けることが可能である。図 9 の例では CFD のデータと風洞実験のデータを独立の階層に格納している。さらに下位の階層において個別の詳細なデータが格納される。各階層が保持するデータは図 10 から図 12 のように表単位・文書単位・画像単位で格納される。目的のデータを探し出す場合には階層構造の根の部分から順にたどっていくこととなる。例えば風洞実験の比較用圧力孔データが必要な場合には ONERA_M5、風試、PIV、比較用圧力孔データの順にたどる。この経路さえ維持されていれば常に同一のデータを取り出すことが可能である。したがって新しいデータを格納する必要が生じた場合でも任意の場所に新しい項目を追加するだけでデータ構造の拡張が完了し、既存のデータ項目には今まで通りアクセス可能である。こうすることで互換性を維持しつつ拡張することが容易となる。

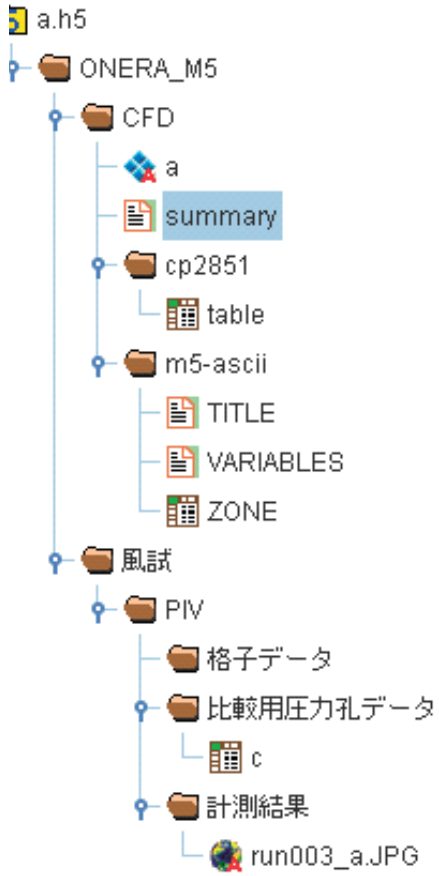


図 9 HDF5 の階層構造

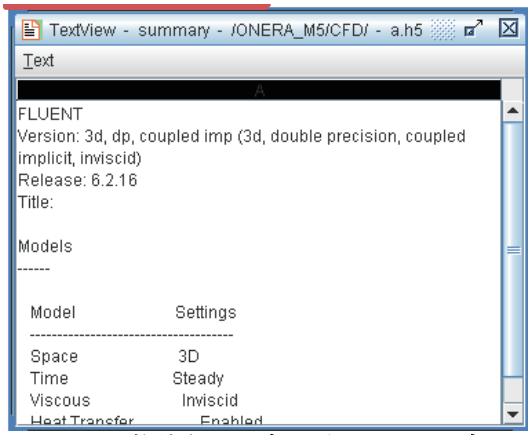
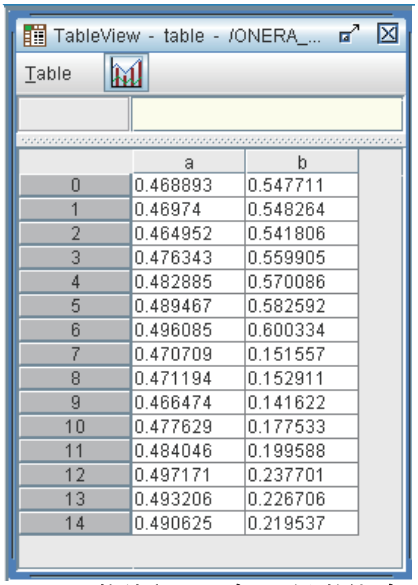


図 10 HDF に格納されるデータ例(テキストデータ)



	a	b
0	0.468893	0.547711
1	0.46974	0.548264
2	0.464952	0.541806
3	0.476343	0.559905
4	0.482885	0.570086
5	0.489467	0.582592
6	0.496085	0.600334
7	0.470709	0.151557
8	0.471194	0.152911
9	0.466474	0.141622
10	0.477629	0.177533
11	0.484046	0.199588
12	0.497171	0.237701
13	0.493206	0.226706
14	0.490625	0.219537

図 11 HDF に格納されるデータ例(数値データ)

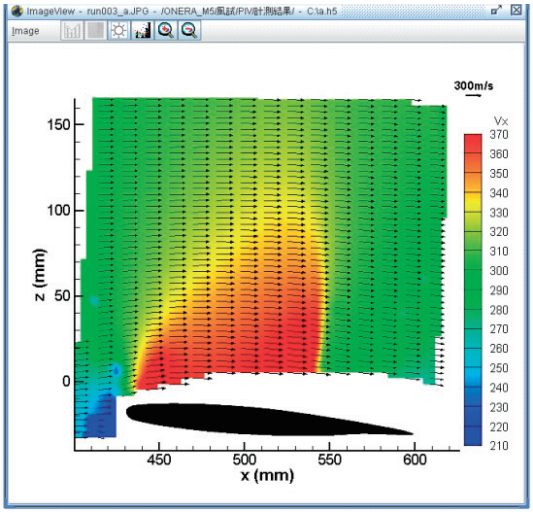


図 12 HDF に格納されるデータ例(画像データ)

(2)HDF 入出力 API を組み込んだプログラム作成

HDF View は手動で操作するため大量のデータの処理には向かないため、大量のデータを処理する場合には別途専用のプログラムを作成する必要がある。

HDF5 フォーマットの開発元が Windows・Linux・Mac 等の各 OS 向けに HDF5 形式でのデータ入出力を行うためのライブラリプログラムを公開しているため、これを自分の

プログラムに組み込むことでHDF5形式による入出力機能を搭載する方法である。市販の可視化ソフトウェア等はこれらのAPIを自社製品に組み込むことでHDF5形式への対応を行っている。Fortran・C等の各言語用APIが公開されているため、大抵のCFDコードへの組み込みは容易である。手元にソースコードがない場合やプログラム本体に手を加えたくない場合には別途これらのAPIを用いた変換プログラムを作ることで対応することも可能である。実際にFortranプログラムにおいてHDF5形式による入出力機能を実装した例を表3に示す。このプログラムでは図13のような2次元配列の書き込みを行っている。大まかにはHDF5の入出力機能の初期化、HDF5形式のファイルを開く、データセット(個々の階層)を開く、データを書き込み、データセット、ファイルを閉じた後に終了処理を行うという流れとなっている。個々の処理はサブルーチンを1つ呼び出すのみとなっており、通常のFortranプログラムによるファイル入出力と手間は大きくは変わらないことがわかる。

表3 Fortranプログラムでの HDF5 の使用例

```
PROGRAM RWDSETEXAMPLE
USE HDF5 ! This module contains all necessary modules

IMPLICIT NONE
CHARACTER(LEN=8), PARAMETER :: filename = “dsetf.h5”
CHARACTER(LEN=4), PARAMETER :: dsetname = “dset”
INTEGER(HID_T) :: file_id       ! File identifier
INTEGER(HID_T) :: dset_id       ! Dataset identifier
INTEGER     ::   error ! Error flag
INTEGER     ::  i, j
INTEGER, DIMENSION(4,6) :: dset_data, data_out ! Data
buffers
INTEGER(HSIZE_T), DIMENSION(2) :: data_dims

do i = 1, 4
    do j = 1, 6
        dset_data(i,j) = (i-1)*6 + j;
    end do
end do

!FORTRAN インターフェースの初期化
CALL h5open_f(error)
!ファイルを開く
CALL h5fopen_f (filename, H5F_ACC_RDWR_F, file_id, error)
!データセットを開く
CALL h5dopen_f(file_id, dsetname, dset_id, error)
data_dims(1) = 4
data_dims(2) = 6
!データ書き込み
CALL h5dwrite_f(dset_id, H5T_NATIVE_INTEGER, dset_data,
data_dims, error)
CALL h5dread_f(dset_id, H5T_NATIVE_INTEGER, data_out,
data_dims, error)
!データセットを閉じる。
CALL h5dclose_f(dset_id, error)
!ファイルを閉じる。
CALL h5fclose_f(file_id, error)
!FORTRAN インターフェースの終了処理
CALL h5close_f(error)
END PROGRAM RWDSETEXAMPLE
```

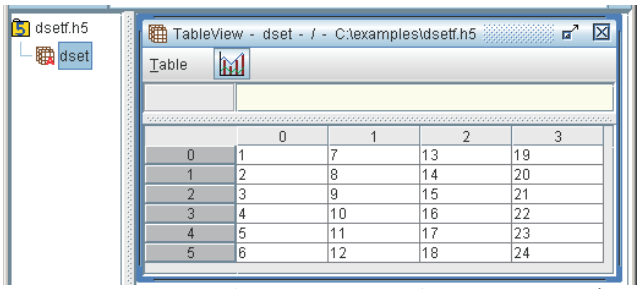


図13 サンプルプログラムによって生成される HDF5 データ

HDF5内部の階層構造の設計には留意すべき点がいくつかある。1つ目は既存のHDF5形式に対応する可視化ソフトウェアを利用する場合にはその可視化ソフトが定義するHDF5形式にあわせて階層構造を作る必要がある。これはHDF5ファイル内部で階層構造をたどっていく順番が変わってしまうと目的のデータが読み出せなくなってしまうからである。したがって最初にどのような可視化ソフトを利用するかを決定し、そのソフトウェアがHDF5形式をサポートする場合にはそのソフトウェアが読み出せる階層構造を定義し、可視化には必要ないその他のデータについては別途階層構造を追加していくという方法をとるのが望ましい。後で項目を追加しても可視化ソフトは無視して必要なデータのみを読み込むため、互換性を維持しつつ、情報を付加していくことが可能となる。後々のデータ抽出が容易になるか煩雑になるかはこのHDF5形式のファイル内部での階層構造をどのように組み立てるかにかかっている。例えばテキストデータを格納する場合には1つのテキストデータを分割して格納することでテキスト内の必要とされる部分のみだけ取り出す、といった方法が考えられる。具体例としては表4のようにテキストデータを3分割してそれぞれを個別の項目として格納する(図 14)ことが考えられる。HDF5では格納するデータがテキストなのか、浮動小数点なのか、整数なのかを識別して格納することが出来るため、数値データが表形式で並べて格納されているような場合にはこの部分のみを浮動小数点の2次元配列として格納してしまうことで、読み込み後の処理の実装も容易となる。また、格子データの補間を行う場合には全格子のデータが必ずしも必要ではないため、分割して格納することで必要な格子データだけ取り出して高速に補間処理を行うといったことも可能である。

表4 テキストファイルの HDF5 への分割格納例

```
TITLE     = "fluent6.2.16"

(ここで分割)

VARIABLES = "X"
"Y"
"Z"
"Pressure"
"X Velocity"
"Y Velocity"
"Z Velocity"
"body-force 0-1"
"body-force 0-2"
省略
DATASETAUXDATA Common.WVar="7"

(ここで分割)

ZONE T="fluid-2"
```

```
 STRANDID=0, SOLUTIONTIME=0
 N=1233349, E=1843032, ZONETYPE=FEBrick
 DATAPACKING=BLOCK
 DT=(DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE
DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE
DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE
DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE )
 0.000000000E+00 0.000000000E+00 0.000000000E+00
0.000000000E+00 2.749077176E+01
 0.000000000E+00 0.000000000E+00 2.749077176E+01
2.749077176E+01 5.498154353E+01
 5.498154353E+01 2.749077176E+01 5.498154353E+01
2.749077176E+01 5.498154353E+01
 5.498154353E+01 5.498154353E+01 2.749077176E+01
8.247231529E+01 8.247231529E+01
 8.247231529E+01 1.099630871E+02 1.099630871E+02
8.247231529E+01 1.099630871E+02
省略
```

図 14 テキストデータの分割格納例

6. HDF5 のメリットと課題

HDF5 のメリットと課題についてまとめる。

HDF5 のメリット

- エンディアンの違いなどを吸収するため使用者は CPU の違いを意識せずに済む
- ファイルの入出力が高速 である。
- 新たなパラメータを追加するなどの拡張が容易 である。

HDF5 そのものの課題

画像ファイルは HDF5 内部では独自形式で保存されるため、HDF5 へ画像を保存するには下記の方法のうちいずれかを行う必要がある。

- HDF View を用いて jpeg ファイルを保存する
- gif2h5 コマンドを使って gif から HDF ファイルを生成する。
- 変換プログラムの自作

HDF View の課題

これは HDF View というソフトウェア自身の実装に由来する課題であり、HDF5 形式そのものの制限に由来するわけではない。

- Undo に未対応であるため、編集結果が即座に HDF5 ファイルへ反映されてしまう。
- Jpeg 形式の画像ファイルのみ取り込み・出力可能 である。他の形式の画像を保存する場合には Jpeg 形式にいったん変換する必要があるため、画質の劣化を伴ってしまう。
- 大きな画像・データの入出力時にメモリ不足に陥る。具体的な数値は不明だが、Java VM の仕様が原因であるため、HDF View が Java で実装されている限りは回避が困難である。
- ファイルパスに日本語名が含まれる場合にはファイルの入出力に失敗する。

運用上の課題

- HDF ファイル内の階層構造とデータをどのように格納するかのルールについては後々にデータの検索・抽出が容易になるように考慮する必要がある。
- 可視化ソフトウェアから直接読み込む場合には可視化ソフトにあわせて階層構造を定義してから、その他の情報を付加する必要がある。

7. 結言

本稿では、JAXA ハイブリッド風洞構想のシステム実現において、最も重要な項目であるシステム構成と基盤となるデータフォーマットについて検討した結果について述べた。今後、JAXA ハイブリッド風洞構想の概念検討および予備設計を実施する中でこれらを詳細化していく予定である。

参考文献

[1]中北、「大型実用風洞における感圧塗料技術の研究開発」、JAXA 第 1 回総合技術研究本部公開研究発表会 前刷集、2004. 3
[2]Richard J. Schwartz, Gary A. Fleming, ICIASF2007
[3]渡辺他、「PIV(粒子画像流速測定法)を用いた CFD コード検証について」、JAXA-SP-04-012、2005. 3
[4] http://www.atmarkit.co.jp/fxml/rensai/rexml01/rexml01.html
[5] http://hdf.ncsa.uiuc.edu/HDF5/: Mike Folk, "HDF and HDF5", HDF/HDF-EOS Workshop III, 1999.9
[6] Goucher, G., and Mathews, J., "The National Space Science Data Center Common Data Format," Research and Technology R&T Report, NASA/Goddard Space Flight Center Publication, December 1994. : http://cdf.gsfc.nasa.gov/
[7] http://hdf.ncsa.uiuc.edu/hdf-java-html/hdfview