JAERI-Data/Code
2001-003

可視化デバッガの改良

ーデータ取得機能の高速化と複数のプログラムの比較、組み合わせ表示機能の開発ー

2001年3月

松田　勝之・武宮　博

日本原子力研究所
Japan Atomic Energy Research Institute

本レポートは，日本原子力研究所が不定期に公刊している研究報告書です。
入手の問合わせは，日本原子力研究所研究情報部研究情報課（〒319-1195　茨城県那珂郡東海村）あて，お申し越しください。なお，このほかに財団法人原子力弘済会資料センター（〒319-1195　茨城県那珂郡東海村日本原子力研究所内）で複写による実費頒布をおこなっております。

This report is issued irregularly.
Inquiries about availability of the reports should be addressed to Research Information Division, Department of Intellectual Resources, Japan Atomic Energy Research Institute, Tokai-mura, Naka-gun, Ibaraki-ken, 319-1195, Japan.

編集兼発行　　日本原子力研究所

JAERI-Data/Code 2001-003

可視化デバッガの改良
—データ取得機能の高速化と複数のプログラムの比較、組み合わせ表示機能の開発—

日本原子力研究所計算科学技術推進センター
松田 勝之・武宮 博※

(2001年1月19日受理)

科学技術計算プログラムのデバッグ作業を支援する可視化デバッガ vdebug におけるデータ取得を高速化した．また，複数のプログラムを比較したり，組み合わせて表示する機能を追加した．データ取得の高速化では，SX-4 の dbx，pdbx で 100 倍以上，SR2201 の ndb で 10 倍以上の成果を得ることができた。複数のプログラムの比較では，SP と ONYX の上で実行したプログラムが各計算ステップでほぼ同じ値を示していることを容易にチェックできることを確認した．本報告書では，この vdebug のデータ取得機能に対して行った改良とその結果，並びに複数プログラムの比較方法と実施例について述べる．

日本原子力研究所：〒153-0061 東京都目黒区中目黒 2-2-54
※ 外来研究員：日立東北ソフトウエア(株)

JAERI-Data/Code 2001-003

Improvement of Visual Debugging Tool — Shortening the Elapsed Time for Getting Data and Adding New Functions to Compare/Combine a Set of Visualized Data —

Katsuyuki MATSUDA and Hiroshi TAKEMIYA※

Center for Promotion of Computational Science and Engineering
Japan Atomic Energy Research Institute
Nakameguro, Meguro-ku, Tokyo

(Received January 19, 2001)

The visual debugging tool "vdebug" has been improved, which was desigend for the debugging of programs for scientific computing. Improved were the following two points; (1)shortening the elapsed time required for getting appropriate data to visualize; (2)adding new functions which enable to compare and/or combine a set of visualized data originated from two or more different programs. As for shortening elapsed time for getting data, with the improved version of "vdebug", we could achieve the following results; over hundred times shortening the elapsed time with dbx, pdbx of SX-4 and over ten times with ndb of SR2201. As for the new functions to compare/combine visualized data, it was confirmed that we could easily checked the consistency between the computational results obtained in each calculational steps on two different computers: SP and ONYX. In this report, we illustrate how the tool "vdebug" has been improved with an example.

Keywords: Visualization, Debugging, Visual Analysis, Scientific Simulation, Debugger

※ On leave from Hitachi Tohoku Software,Ltd

目　次

Contents

1 はじめに

科学技術計算におけるデバッグ作業では、大規模な配列の値を確認する必要があるため、変数の値を数値だけで表示するデバッガは膨大な数字テーブルを検証することは作業効率が悪くあまり使用されてこなかった。そこで、デバッガを用いてプログラムの実行を制御し、プログラム上の任意の位置で任意の配列データを取得して可視化を行うことでデバッグ作業を効率的に支援する可視化デバッガ[1]-[4]を開発してきた。近年、TotalView[5]、PSUITE[6]といった配列データの可視化を行う機能を持ったデバッガも提供されるようになってきたが、これらのデバッガは配列データを正規空間上で可視化するだけで、物理空間へ写像した上での可視化には対応していない。デバッグ中に物理量としての可視化を行うためには、可視化ツールとデバッガをうまく連携させる必要がある。

可視化デバッガはプログラム上の任意の位置にブレークポイントを設定し、その位置でユーザの指定したデータを取得することができる。また、データの取得後プログラムの実行の継続コマンドを自動的に発行する。そのため、

(1) プログラムをリコンパイルすることなく任意の位置での任意のデータをフレキシブルに可視化できる。

(2) データの取得位置をループの内側に設定した場合、簡易なトラッキングツールの役割を果たすことができる。

といった利点がある。

プログラムのモデルやパラメタを変えたり、プログラムを並列化したり他のマシンに移植した場合など、オリジナルの実行結果と変更あるいは移植後の結果を比較することはしばしば必要とされる。特に、並列化や移植などの作業中に任意のデータを作業の前後で簡便に比較できれば作業の効率を高めることができる。また、大量のデータを比較するためにはデータの差分のみを可視化することは有効な手段となる。

今回、複数のアプリケーションプログラムに対するデバッグ機能を追加し、それぞれのプログラムから取得したデータの差分を可視化したり、組み合わせて可視化することができるようにした。これにより、容易に複数のプログラムの比較が行えるようになった。

本報告書は、この可視化デバッガにおける複数プログラムのデバッグ機能の開発とデータ取得の高速化のための改良に関してまとめたものである。2章では並列可視化デバッガの機能とシステム構成について、3章ではデータ取得の高速化について、4章では複数のプログラムのデータ比較について、5章では2つのプログラムの結果の差分表示の実行例について記述している。

2 可視化デバッガ vdebug の特徴

可視化デバッガは、並列デバッガと可視化アプリケーションを連携させたシステムである。デバッガを通してアプリケーションを実行することでプログラムの実行をユーザの指定した任意の位置で中断することができる。中断された時点でデバッガから参照できる任意の配列変数をデバッガの機能を用いて取得し、これを可視化アプリケーションの利用できるフォーマットに変換した上で可視化アプリケーションに受け渡す。さらに、このまま実行を継続することで、簡易なトラッキング機能を果たすことができる。このようにして可視化デバッガではアプリケーションの任意の位置で任意のデータを可視化するので、可視化するデータやそのデータを取得する位置をアプリケーションを再コンパイルすることなく柔軟に指定することができる。

可視化方法、データの対応付け等は可視化テンプレートとして用意し、ユーザは任意の可視化テンプレートを利用することができる。複数の入力モジュールを扱うような可視化テンプレートを用意することで、複数のアプリケーションの比較をしたり、組み合わせて表示したりすることもできる。

本章では、可視化デバッガのシステム構成、データの取得方法、複数のアプリケーションを比較・組み合わせる方法の概略について述べる。

2.1 システム構成

並列可視化デバッガは、全体としてユーザインターフェイス部、デバッガサーバ、デバッガ、データフォーマットサブシステム、データマージサブシステム、可視化サブシステムの 6 つのサブシステムから構成される。各サブシステムの機能を表 2.1 に示し、サブシステム間の関係を図 2.1 に示す。

表 2.1各サブシステムの機能

サブシステム名称	プログラム名称	機　能
ユーザ インターフェイス部	vdebug	並列可視化デバッガのユーザインターフェイス。ユーザの要求を dbserver に通信し、dbserver からデータを得、avs_client を起動して可視化する。
	Template	vdebug で得た変数の値を AVS で表示するための手順を記述したファイル。vdebug の中で作成することもできる。cli ファイルや formater ファイルの名称や変数の型などを記述する。
	Filter	vdebug で得た各ノードの局所データを大域データに組み上げるためのデータの分散方法を記述したファイル。vdebug の中で作成することもできる。
	avs_client	サーバモードで起動された AVS と通信するクライアント。vdebug から AVS へ渡すメッセージを標準入力から受け取る。
デバッガサーバ	dbesrver	vdebug と socket 通信を行い、子プロセスとしてデバッガを起動する。デバッガとはパイプまたは擬似端末を利用して通信する。
デバッガ		各マシンのデバッガ。現在 gdb,dbx,ladebug,fdb pdbx,ndb,totalview,pfdb,ipd に対応している。
データマージ サブシステム	merger	vdebug で得た各ノードの局所データを大域データに組み上げるマージプログラム。
データフォーマット サブシステム	formater	vdebug で得た変数を AVS が読み込める形式に変換するための整形プログラム。
可視化サブシステム	cli	AVS のネットワークやパラメータを記述したファイル。
	AVS	さまざまな表現のできる可視化システム。サーバモードで起動する。

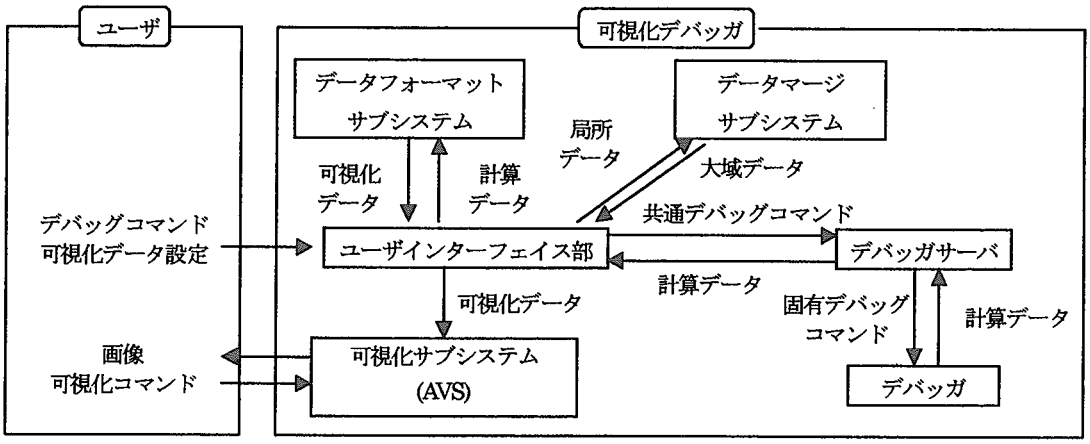

図 2.1システム構成

ユーザがユーザインターフェイス部を操作すると、それに対応したプロトコル(共通デバッグコマンド)がデバッガサーバに送られ、デバッガサーバは各デバッガの固有のコマンドを実行する。そして、デバッガの出力したメッセージをデバッガサーバが解析してユーザインターフェイス部に返信する。可視化すべきデータが得られると、ユーザインターフェイス部は指定に応じて、データマージサブシステムにより大域データを組み上げ、データフォーマットサブシステムにより可視化サブシステムが読みこめる形式に変換して可視化サブシステムに送信する。可視化サブシステムはこのデータを表示する。

vdebug を起動すると図 2.2 に示すようなユーザインタフェイス画面が表示される。

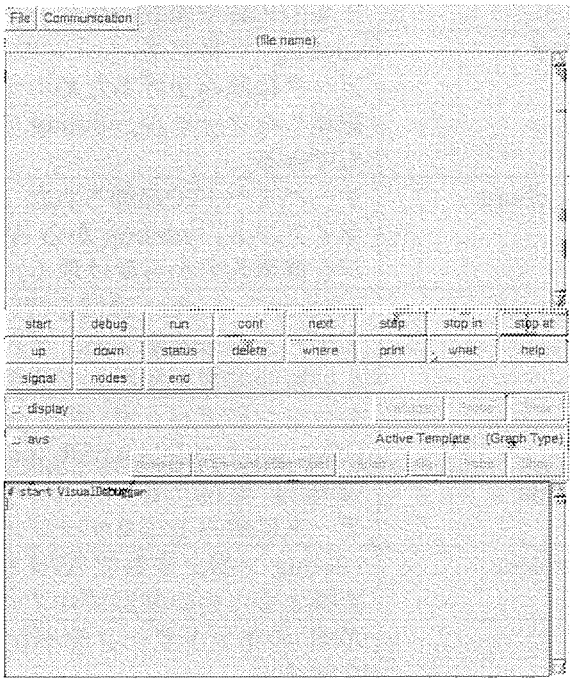

図 2.2vdebug 起動直後の画面

2.2 デバッガを通してのアプリケーションからのデータ取得

可視化デバッガは、アプリケーションプログラムの任意の位置で任意の配列データを取得し、そのデータを可視化することを目的としている。そのためデバッグの対象となる大量のデータをデバッガを通して高速に読み込む必要がある。

デバッガから配列データを取得する手段として一般的な方法は print コマンドによって出力することであろう。しかし、この方法は大量のデータを出力する上では遅すぎる。そのため筆者らは、表 2.2 に示す方法を検討した。

表 2.2 デバッガを通してデータを取得する方法

番号	方　法
1	動的関数呼出コマンドを利用して write システムコールを呼び出す。
2	動的関数呼出コマンドを利用してデータを出力するための可視化デバッガライブラリを呼び出す。
3	あらかじめシグナルハンドラにデータを出力するための可視化デバッガライブラリを登録しておき、シグナルを通知することでライブラリを呼び出す。
4	デバッガを改良してバイナリ出力のコマンドを追加する。

利用するデバッガによりどの手段をとるべきかは異なるが、デバッガに対し容易に改良が施せるのであれば、4 の方法が一番簡単なアプローチであろう。ついで動的関数呼出コマンドが使えれば 1 の方法がデバッグ対象のアプリケーションに手を加える必要がないので望ましいと思われる。2 の方法はデバッガがシステムコールを呼び出せない場合、あるいはシステムコールの呼び出しに時間がかかり、データをまとめて出力させるような工夫が必要な場合に有効であろう。3 の方法は動的関数呼出が行えない、またはそのコマンドが実用的な速度で動作しない場合に対する対策となる。

2.3 複数のプログラムの比較・組み合わせ

プログラムのモデルやパラメタを変えたり、プログラムを並列化したり他のマシンに移植した場合など、オリジナルの実行結果と変更あるいは移植後の結果を比較することが必要とされる場面はいろいろ考えられる。このような比較を可視化デバッガを用いて行えるように、複数のアプリケーションプログラムを制御できるよう改良した。これにより、比較するだけでなく組み合わせての表示も行える。

可視化デバッガは、デバッガサーバがデバッガを通してプログラムの実行を制御しデータを取得する。複数のプログラムを制御して得られたデータを比較あるいは組み合わせるためには複数のデバッガサーバを用いてそれぞれコントロールする必要がある。さらに、それぞれから得られたデータを区別して可視化サブシステムに送付する必要がある。

そのため、デバッガサーバとの通信を行うソケットを制御するデバッガサーバの数だけ作成しこれをユーザインターフェイス部で配列として管理することとした。また、2 つ目以降のデバッガサーバと接続するに当たって、ウィンドウの複製と同様のやり方でウインドウを主画面を作成し、これと対応付けてソケットの情報を管理する。このソケットとウインドウの配列をもとに、イベントを管理する。すなわち、それぞれのウィンドウ上でのユーザの操作は対応するソケットを通してそれぞれのデバッガサーバへメッセージが送られる。そして各ウインドウのイベントループの中でそれぞれのデバッガサーバからのメッセージのチェックが行われる。

複数のデバッガサーバから得られるデータを区別して可視化サブシステムに送付するためには、1 つのモジュールネットワークに複数の入力モジュールを持ち、それぞれのデータを比較あるいは組み合わせるような CLI を作成し、この CLI を利用する可視化テンプレートをそれぞれの入力モジュールを用いるように必要なだけ用意する。各ウインドウで

は対応する入力モジュールを利用するテンプレートを指定し、これによりデータの比較あるいは組み合わせを行う。

この様子を図 2.3 に示す。

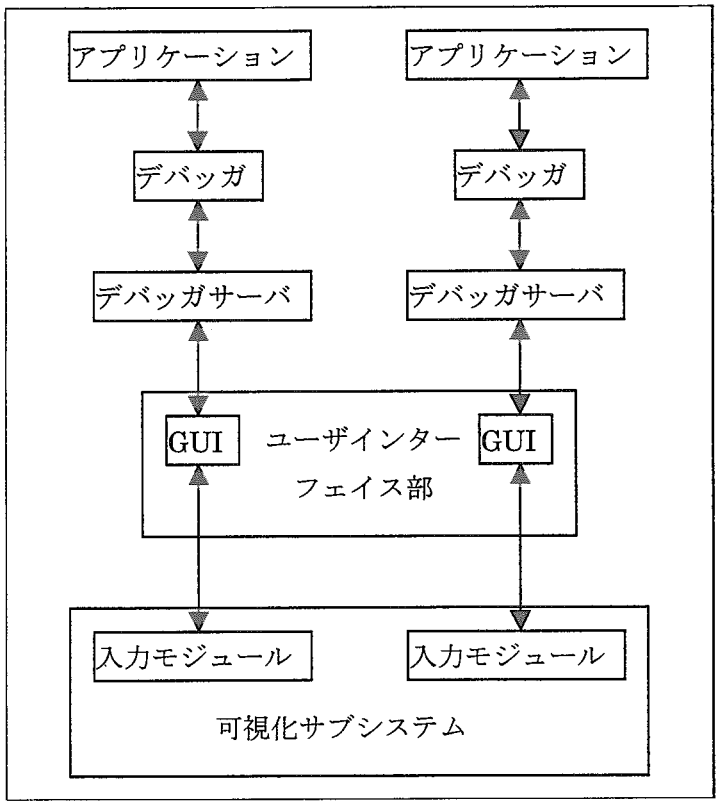

図 2.3 複数のアプリケーションプログラムを比較・組み合わせる概要

3 データ取得の高速化

本章では、表 2.2 に示したデータの取得方法について議論する。

3.1 動的関数呼出コマンドによるシステムコール呼出

表 2.2 に示した 1 の方法である。この方法が実用的な速度で用いられるのであれば、デバッグ対象のアプリケーションに対して修正を加えることなくデバッグが行える。データ取得位置をブレークポイントとして設定しておき、このブレークポイントに到達したことを示すメッセージがデバッガから届いたら、動的関数呼出コマンドを用いて write システムコールにより FIFO へデータを出力する。FIFO はデバッガを exec する前にあらかじめオープンしておく。デバッガによっては起動時にオープンしてあるファイルをクローズするようになっているものもある。その場合には関数の戻り値を返す動的関数呼出コマンドを用いて open システムコールを呼び出し FIFO をオープンする。

具体的なアプローチを図 3.1 に示す。

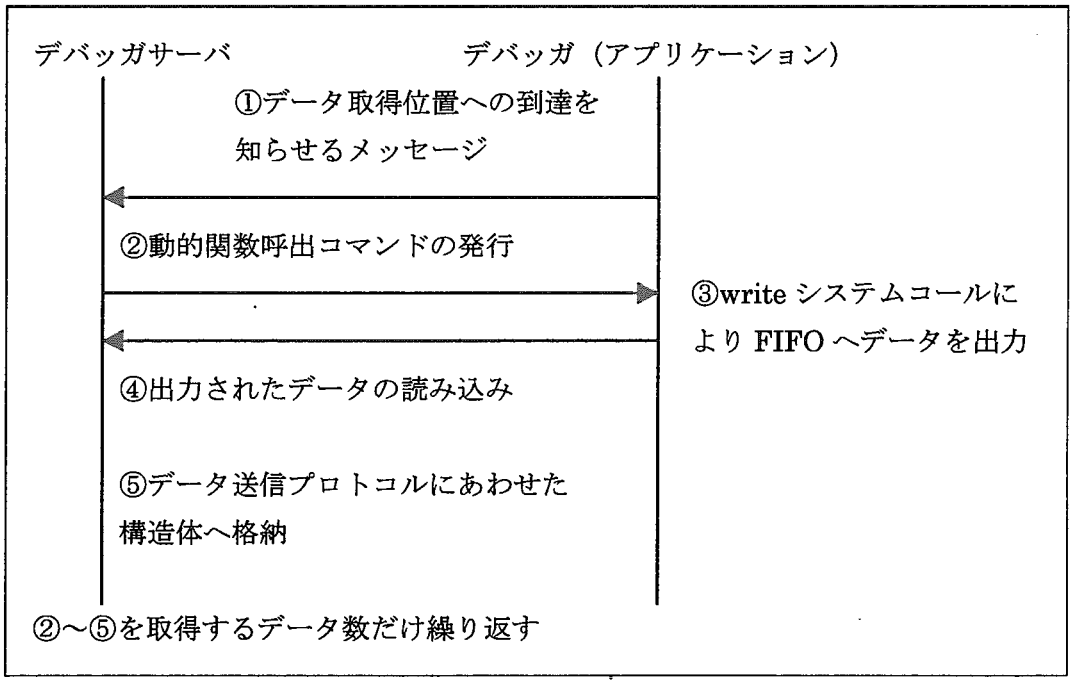

図 3.1 動的関数呼出コマンドを用いたデータ取得

この方法により表 3.1 に示すマシンで動作を確認している。

表 3.1 動的関数呼出コマンドによりシステムコールを呼び出しができる機種

WS・Super Computer	OS	デバッガ名
SUN	Solaris2.6	dbx[7]
		gdb[8]
SGI	IRIX6.2	dbx[9]
Compaq Tru64 UNIX	Digital UNIX V4.0E	dbx[10]
IBM SP	AIX3.4	dbx[11]
		pdbx[12]
NEC SX-4	SUPER-UX 8.1	dbx[13]
		pdbx[14]
CRAY T94	UNICOS 10.0.0.3	totalview[15]

この方法の特徴をまとめると以下のようになる。

長所

- システムコールを呼び出すのでデバッグ対象のアプリケーションに手を加える必要がない。

短所

- write システムコールを用いるので取得対象となるデータが多数ある場合、何度も呼び出す必要がある。

IBM SP の pdbx[12]の場合、動的関数呼出コマンドの実行に際し比較的時間を要する。例えば 3 次元の座標値、3 方向の流速データといったデータを取得する場合、6 回の呼び出しを行う必要がある。また、事前に FIFO をオープンしておけないので内部でオープンする必要があり、そうすると必ずしも各ノードで FIFO のファイルディスクリプタが一致するとは限らないので、ノード数分呼び出す必要もあるので 10 並列であれば 60 回の呼出になり、数十秒の時間がかかる。そのため、複数のデータをまとめて出力する関数を作成し、それを呼び出すように改良した。

NEC SX-4 の dbx[13]および pdbx[14]でも動的関数呼び出しは行えるのだが、1 回の呼び出しに数分程度かかり実用的でない。そのため、シグナルを用いるように改良した。

3.2 動的関数呼出コマンドによる可視化デバッガライブラリ関数呼出

表 2.2 に示した 2 の方法である。この方法では、デバッグ対象のアプリケーションに対して可視化デバッガのライブラリをリンクする必要はあるが、可変個数引数リストを持つ関数の動的呼出が可能であれば、データをまとめて出力するようにするなどの改良が行える。また、システムコールを直接呼び出せないようなデバッガでもデバッグが行える。

データを取得する手順は基本的にシステムコールの場合と同様である。ただし可変個数引数リストを持つ関数の動的呼出が可能であれば、取得すべきデータを引数に並べて呼び出せるので、動的関数呼出コマンドは一度だけ実行すればよい。

可変個数引数リストを利用した出力関数を利用したアプローチを図 3.2 に示す。

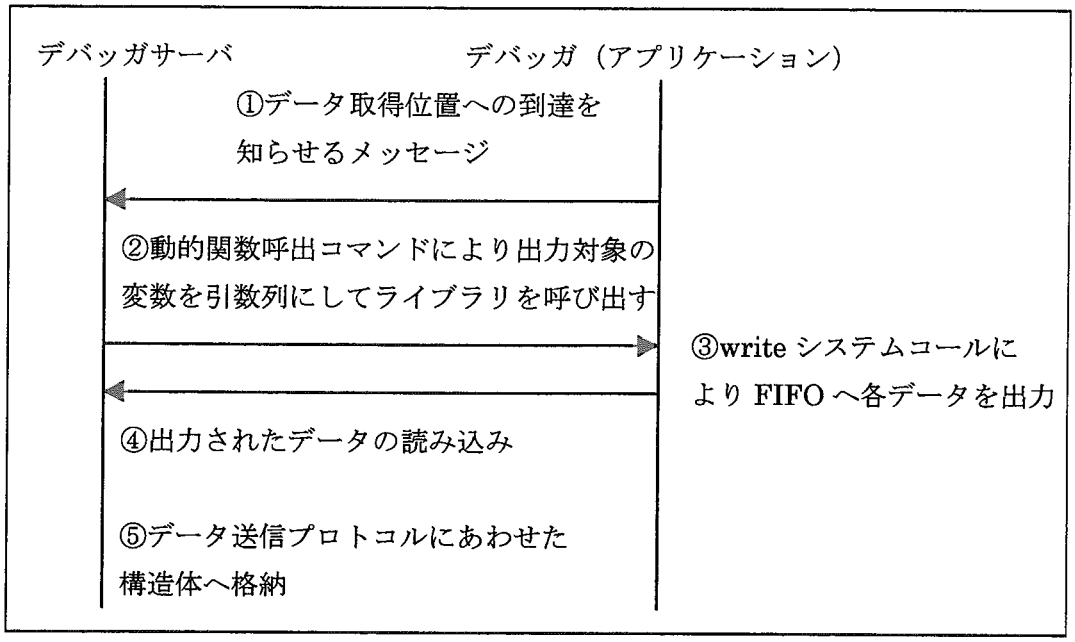

図 3.2 動的関数呼出コマンドを用いたデータ取得

この方法により表 3.2 に示すマシンで動作を確認している。

表 3.2 動的関数呼出コマンドにより可視化デバッガのライブラリを呼び出している機種

WS・Super Computer	OS	デバッガ名	可変個数引数リスト
Compaq Tru64 UNIX	Digital UNIX V4.0E	ladebug[16]	使用不可
IBM SP	AIX3.4	pdbx[12]	使用可能

Compaq/Tru64 UNIX の ladebug[16]の場合、システムコールを直接呼び出すことはできないが静的にリンクされている関数は呼び出すことができる。ただし、可変個数引数リストを持つ関数を呼び出すことはできないので、単純に write システムコールを呼び出す関数を作成して動作を確認した。Compaq/Tru64 UNIX の場合、ライブラリを利用するためにはオブジェクトファイルをリンクするだけでよいので、デバッグ対象アプリケーションのソースファイルを変更する必要はない。

IBM SP の pdbx[12]の場合、可変個数引数リストを持つ関数を呼び出すことができるので、各ノードごとのファイルディスクリプタならびに出力対象変数とそのサイズをリストにして受け取る関数を作成し動作を確認した。これにより 50x50 の配列で表わされる X、Y 座標値、およびスカラーデータ 1 つを 2 ノードから取得する場合、1 の方法では 6 回の呼び出しが必要だったのが 1 回でよくなり、データの取得に要する時間がおよそ 1/3 になった。IBM SP の場合、オブジェクトファイルをリンクするだけではその関数は呼び出せないので、ダミーの関数を呼び出すようにデバッグ対象アプリケーションのソースに手を加える必要がある。

3.3 シグナルハンドラの利用

表 2.2 に示した 3 の方法である。この方法では、デバッグ対象のアプリケーションに対して最初にシグナルハンドラの設定を行う可視化デバッガのライブラリ関数を呼び出すようにソースを変更する必要がある。データを取得する手順を図 3.3 に示す。

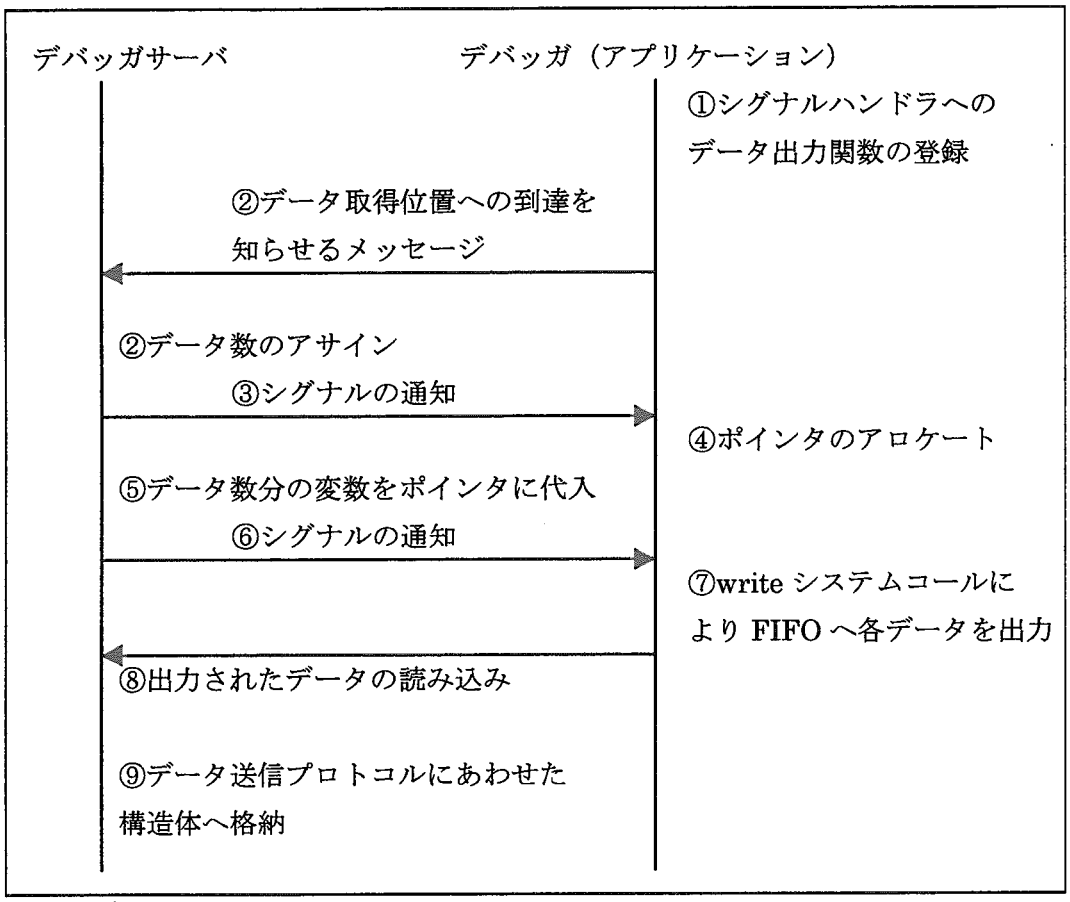

図 3.3 シグナルハンドラを用いたデータ取得

この方法は、NEC SX-4 で dbx[13]、pdbx[14]について確認した。NEC SX-4 では 3.1 で述べたように動的関数呼出コマンドの実行に分単位の時間がかかる。そのためシグナルを利用した処理に改良してみた。その結果、3MB 程度のデータを転送する逐次プログラムに関して 0.2 秒程度に、並列プログラムに関してはノード当たり 0.5 秒程度の時間でデータを取得できるようになった。

この方法の短所としてデバッグ対象アプリケーションのソースに手を入れる必要があることと、デバッグ対象アプリケーションがもともとシグナルを利用していた場合にうまくいかない可能性があることの 2 点が上げられる。

3.4 デバッガへのバイナリ出力コマンドの追加

表 2.2 に示した 4 の方法である。動作としては図 3.1 と同様な手順になり、アプリケーションの実行がデータ取得位置に到達したところで、バイナリ出力コマンドを実行することでデータを取得できる。我々は HITACHI SR2201 の ndb[16]に対し、バイナリ出力コマンドを追加し、テストを行った。その結果、print コマンドにより文字列で出力するのに比べ 10～20 倍の高速化に成功した。

また、デバッガそのものを改良するわけではないが、TotalView V4.0[5]から提供されるようになったコマンドラインインーフェイス(CLI)では、ブレークポイントを指定する際に到達時に実行するコードを指定することができる。これにより、ブレークポイントに到達するたびにデータの出力をデバッガに対して指示する必要がなくなる。この方法での手順を図 3.4 に示す

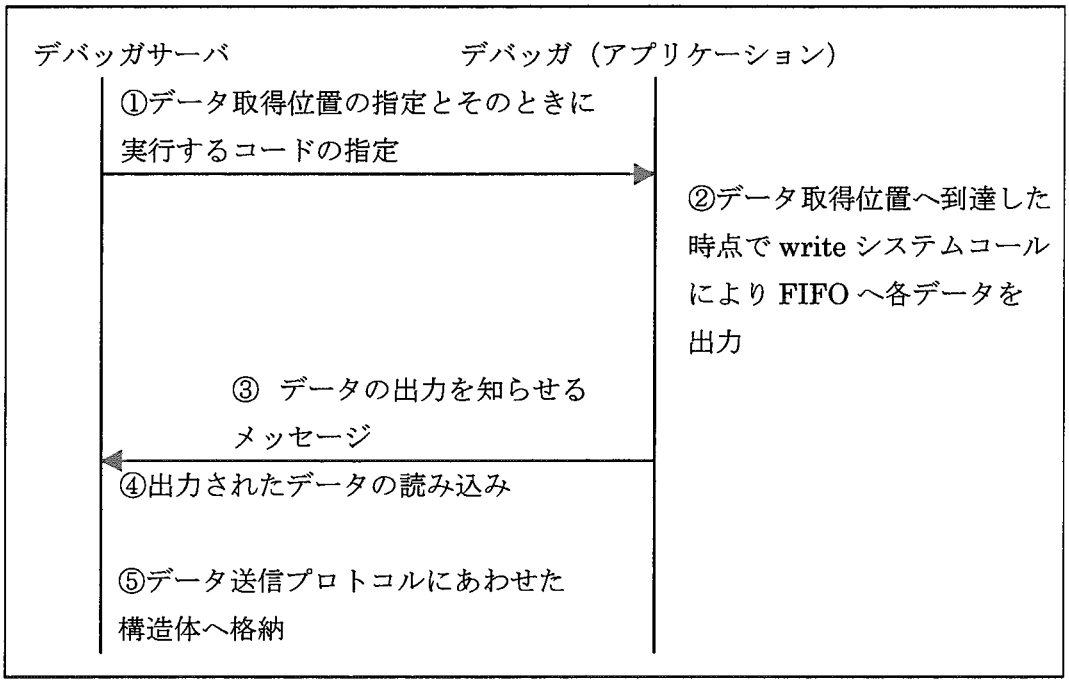

図 3.4 ブレークポイント指定時の実行コード指定を用いたデータ取得

3.5 各方法でのテスト結果

図 3.5～7 に示す翼回りの 2 次元流体解析を表示するためのデータを取得するのに要した時間を表 3.3 に示す。

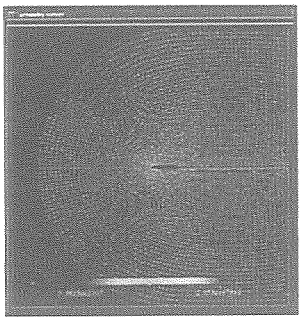

図 3.5 sw2d、psw2d(1)の表示

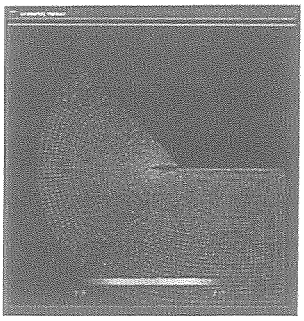

図 3.6 psw2d(2)の表示

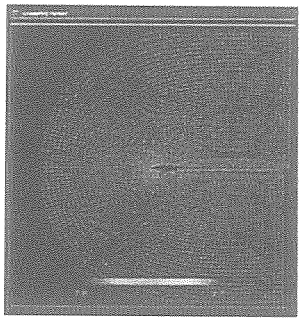

図 3.7 psw2d(3)の表示

sw2d は逐次のアプリケーションでこの表示では xmesh(-2:312,-2:202)、ymesh(-2:312,-2:202)、qq(-2:312,-2:202,1:4)という 3 つの 4 バイト実数型配列の 0:150,1:50,1:1 の部分配列を取得している。psw2d はこれを並列に動かして、各ノードにデータを分散させたもので pxm(1:50,1:50)、pym(1:50,1:50)、pqq(1:50,1:50)という配列にランク 0 では xmesh、ymesh、qq のそれぞれ 1:50,1:50 を、ランク 1 では 51:100,1:50 を、ランク 2 では 101:50,1:50 を代入したものである。psw2d(1)では sw2d と同様に xmesh、ymesh、qq の部分配列を取得、psw2d(2)では pxm、pym、pqq をノード 0、ノード 1 から取得してマージ、psw2d(3)では pxm、pym、pqq をノード 0、ノード 1、ノード 2 から取得してマージしている。

各デバッガでバイナリでデータを取得しているものは、配列の先頭からすべての値をデバッガサーバで読み込んで部分配列をデバッガサーバで切り出している。すなわち、デバッガから取得するデータは xmesh、ymesh はそれぞれ 258KB、qq は 1MB、pxm、pym、pqq はそれぞれ 10KB のサイズである。HITACHI SR2201 の ndb で文字列で出力させているのは、必要部分だけの部分配列である。

計測した時間は、デバッガへデータの取得の指示を出す直前からデバッガサーバが取得したデータを読み終えるまでの時間で、times()関数を用いて計測している。その他込みの時間はデータ取得位置から次のデータ取得位置までの時間であり、データの取得の他にアプリケーションの 1 ステップ分の実行時間を含んでいる。

1 項目分の取得時間が出ているものは各配列データごとの取得時間を平均したものであり、表示されていないものはまとめて取得していることを意味する。測定値は 10～20 ステップ程度実行した結果を平均したものである.

表 3.3 デバッガサーバでデータ取得にかかった時間

マシン	デバッガ名	プログラム	1項目	1枚分	その他込み
SUN	dbx[7]	sw2d	0.01s	0.03s	1.28s
	gdb[8]	sw2d	0.01s	0.03s	0.80s
SGI	dbx[9]	sw2d	0.06s	0.19s	1.21s
Compaq Tru64 UNIX	dbx[10]	sw2d	0.01s	0.03s	0.90s
	ladebug[16]	sw2d	0.02s	0.05s	0.92s
IBM SP	dbx[11]	sw2d	0.02s	0.05s	0.69s
システムコール利用	pdbx[12]	psw2d(1)	0.22s	0.66s	2.05s
		psw2d(2)	0.22s	1.30s	4.17s
IBM SP	pdbx[12]	psw2d(1)		0.23s	1.2s
ライブラリ利用		psw2d(2)		0.44s	2.03s
NEC SX-4	dbx[1][13]	sw2d	23.51s[2]	70.53s[3]	146.38s
動的関数呼出利用	pdbx[4][14]	psw2d(1)	28.01s	84.04s	88.77s
		psw2d(2)	27.63s	165.79s	173.12s
		psw2d(3)	37.63s	338.68s	353.47s
NEC SX-4	dbx[13]	sw2d		0.22s	2.47s
シグナル利用	pdbx[14]	sw2d		0.22s	2.46s
		psw2d(1)		0.57s	3.02s
		psw2d(2)		1.17s	3.48s
		psw2d(3)		1.69s	4.93s
CRAY T94	totalview [15]	sw2d	0.02s	0.07s	1.90s
		psw2d(1)	0.02s	0.07s	4.10s
		psw2d(2)	0.02s	0.14s	1.80s
		psw2d(3)	0.02s	0.20s	2.52s
HITACHI SR2201	ndb[16]	sw2d	45.07s	135.21s	138.96s
文字列出力		psw2d(1)	41.02s	123.05s	123.19s
		psw2d(2)	14.50s	86.99s	90.69s
		psw2d(3)	16.23s	146.11s	151.07s
HITACHI SR2201	ndb[16]	sw2d	2.22s	6.66s	9.79s
バイナリ出力		psw2d(1)	1.77s	5.32s	7.99s
		psw2d(2)	1.51s	9.05s	13.24
		psw2d(3)	1.16s	10.4s	15.59s

この結果を見るとNEC SX-4やHITACHI SR2201では、明らかな改良の成果がでている。IBM SPについては取得する項目数が増えると効果が大きくなることが伺える。これは可変個数引数リストをもつ関数を作成し、動的関数呼出コマンドを 1 度しか呼ばないようにしたことによる成果であろう。しかし、ノード数が増えるとノード数分かかってしまうのは、デバッガの中でノードごとに処理が行われるためであろう。

---

[1] システムコール呼出。

[2] デバッガサーバがデータを読み終えるまでの時間以外に、call コマンドの後処理の時間が 24.56 秒かかる。

[3] call コマンドの後処理の時間を含めると 144.22 秒。

[4] 引数を持たないようにしたライブラリ呼出。SX での動的関数呼出は引数があると余計に時間がかかることがわかったため、とりあえずの改善のためデータは assign コマンドで代入し、引数のない関数を呼び出すこととして対処してみた結果。

4 複数のプログラムの制御

本章では、複数のアプリケーションプログラムの比較・組み合わせを行うためのデバッガサーバと可視化サブシステムの制御について示す。

4.1 複数のデバッガサーバの制御

可視化デバッガでは、ユーザインターフェイス部を操作することによるユーザからの指示とデバッガサーバを介してのデバッガからのメッセージが非同期に届くので、Tcl/Tkの after コマンドを利用してイベントループの中でデバッガからのメッセージがあるかをチェックしている。2.3 で述べたように複数のアプリケーションを比較・組み合わせるためにはそれぞれのアプリケーションを独立したデバッガサーバに制御されるデバッガから起動する必要があり、それぞれのデバッガサーバからのメッセージを区別しなければならない。

本節では、まずユーザインターフェイス部とデバッガサーバの接続方法について述べ、次いで複数のデバッガサーバからのメッセージの区別について述べる。

4.1.1 ユーザインターフェイス部とデバッガサーバとの接続

デバッガサーバは、ユーザインターフェイス部から起動されると、ソケットを生成しポート番号を標準出力に出力する。ユーザインターフェイス部はこのポートに接続するソケットを生成し、そのファイルディスクリプタを保持する。ユーザインターフェイス部とデバッガサーバはこのソケットを通じてメッセージを交換する。デバッガサーバは、ユーザインターフェイス部からのデバッガ指定のメッセージを受けて、子プロセスを作成し、指定されたデバッガを実行する。この手順を図 4.1 に示す。

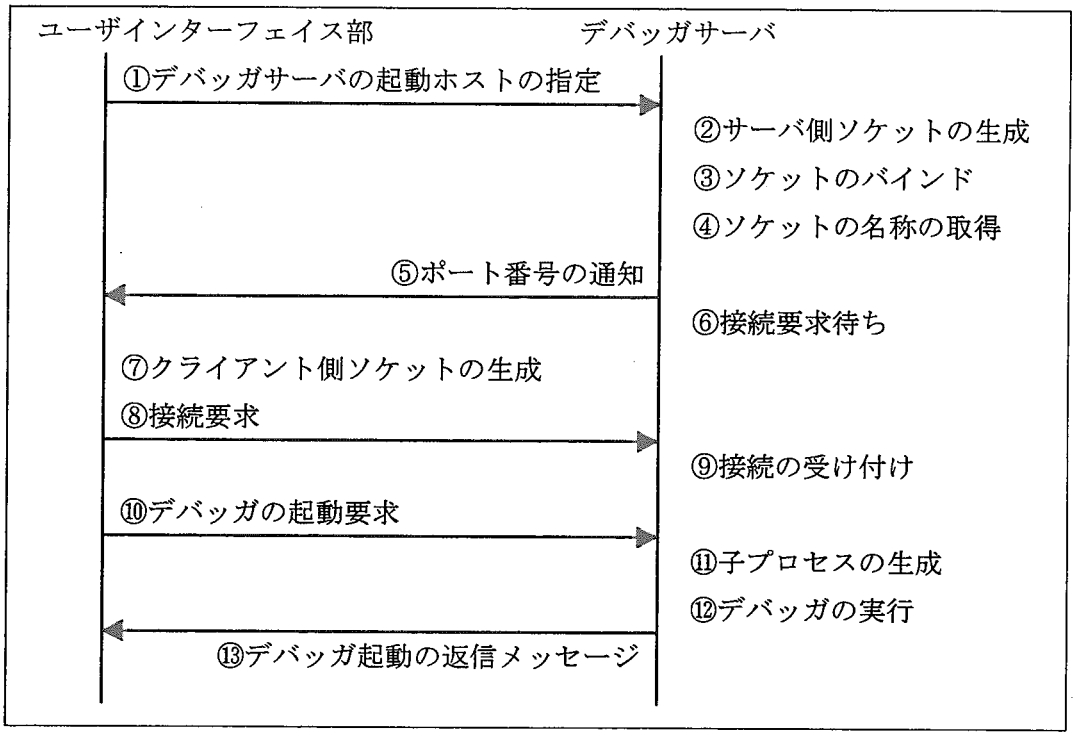

図 4.1 ユーザインターフェイス部とデバッガサーバとの接続手順

デバッガの起動に成功すると、デバッガサーバはユーザインターフェイス部からのメッセージは対象となるデバッガのコマンドに変換してデバッガに、デバッガからのメッセージはプロトコルに則ったメッセージに変換してユーザインターフェイス部に受け渡す。

4.1.2 デバッガサーバの区別

複数のデバッガサーバを起動するときには、ユーザインターフェイス部は対応するGUIを新たに作成して4.1.1に示した手順でデバッガサーバに接続する。このとき得られるクライアント側ソケットのファイルディスクリプタによって、デバッガサーバを区別することができる。このクライアント側ソケットのファイルディスクリプタを以後ソケットIDと呼ぶ。各デバッガサーバはそれぞれ別のGUIと接続されているので、ウインドウIDをキーとしてソケットIDを求め、それに基いてメッセージのチェックを行う。

また、各デバッガサーバのメッセージを順序良くチェックするように、イベントを管理する必要がある。そのため、デバッガサーバからのメッセージをチェックした後に同じデバッガサーバのメッセージをチェックしにいくイベントがあればいったんキャンセルし、再度メッセージをチェックするプロシージャの実行を設定する。このイベント管理のアルゴリズムを図4.2に示す。

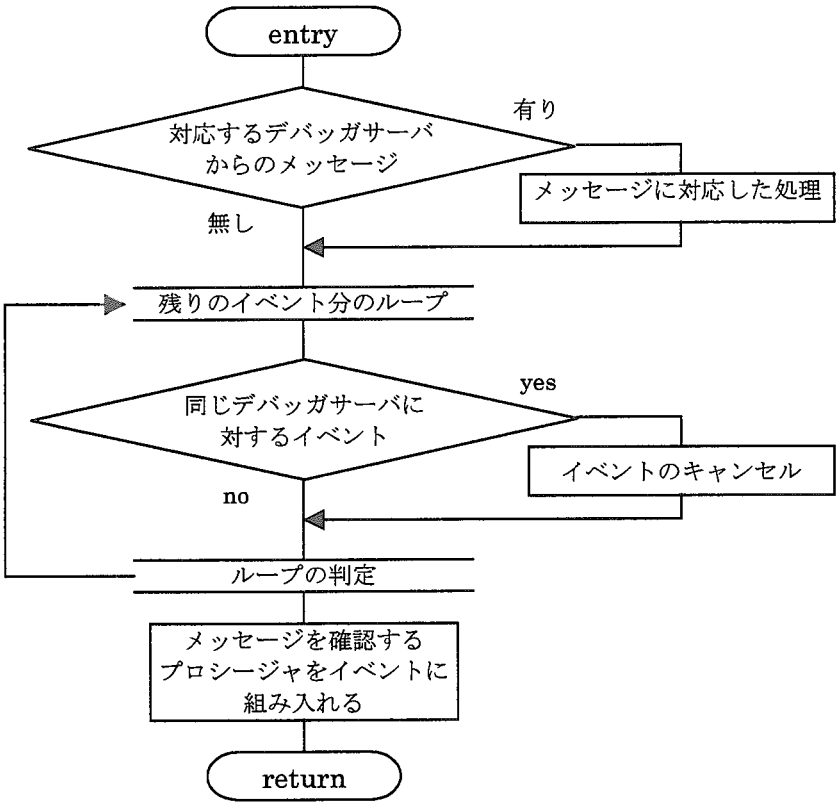

図 4.2 デバッガサーバからのメッセージをチェックする手順

このように処理することにより、GUI の操作に対応したメッセージを各デバッガサーバに伝えるとともに、GUI の動作がアイドル状態になったときに各デバッガサーバからのメッセージの有無をチェックし、対応する処理を行って GUI に表示することができる。

4.2 複数のデバッガサーバから得たデータの整理

複数のデバッガサーバを通して得られたデータを比較あるいは組み合わせて表示するためには、可視化サブシステム側で扱うアプリケーションプログラム数だけ入力モジュールを揃えたモジュールネットワークを用意し、それぞれのデータを区別して入力モジュールに渡してやるようにする。

本節では、まずユーザインターフェイス部と可視化サブシステムの接続方法について述べ、次いでデータと入力モジュールの区別について述べる.

4.2.1 ユーザインターフェイス部と可視化サブシステムとの接続

ユーザインターフェイス部と可視化サブシステムの間ではコマンドの受け渡しとデータの受け渡しの 2 種類の接続がある。データの受け渡しは、ソケットを利用して行っている。ユーザインターフェイス部がサーバ側のソケットを作成し、可視化サブシステムから接続する。そして、ユーザインターフェイス部で接続を受け付け、そのファイルディスクリプタにデータを書き込む。データを書き込む直前に、可視化サブシステムにデータ入力

の準備を促し、このメッセージを受け取って可視化サブシステムはデータの入力を行う。データ受け渡し用のソケットの作成の手順を図 4.3 に示す。

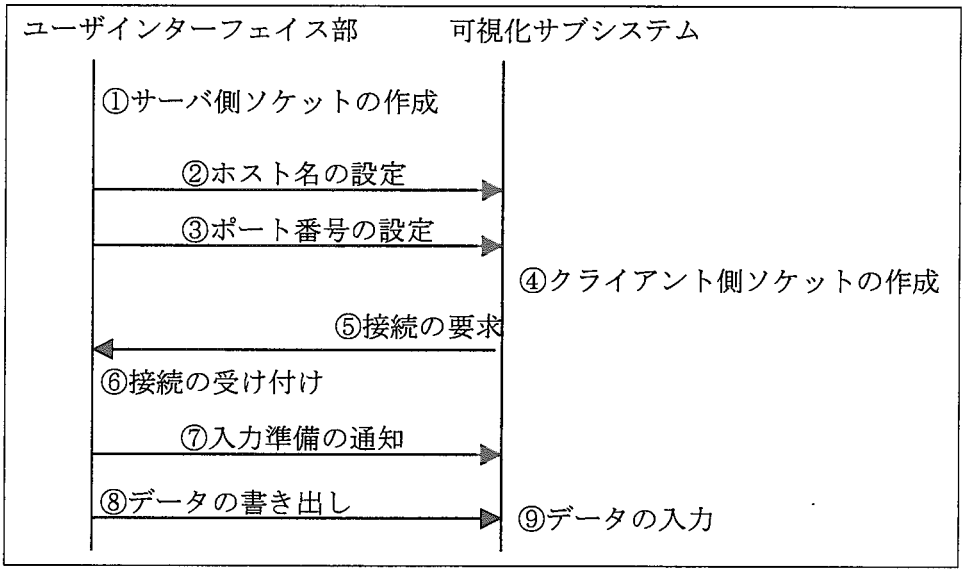

図 4.3 データ受け渡し用ソケットの作成手順

ユーザインターフェイス部から可視化サブシステムに受け渡されるコマンドは、表 4.1 に示すとおりである。このコマンドの受け渡し方法は利用する可視化システムに応じて変わってくる。詳細は各可視化システムのマニュアル[17]-[24]を参照していただきたい。また、AVS を利用する場合については、データ可視化機能を持つ並列プログラムデバッグツール:vdebug—vdebug プログラム設計書—[4]の 5 章も参照していただきたい。

表 4.1 可視化サブシステムに受け渡すコマンド

コマンド	内容	AVS でのコマンド
ネットワーク指定	モジュールネットワークの指定	script –play スクリプトファイル名
ホストアドレス設定	サーバ側ソケットのホストアドレスの指定	parm_set 設定対象パラメタ = 値
ポート番号設定	サーバ側ソケットのポート番号の指定	parm_set 設定対象パラメタ = 値
データ入力指示	ソケットからデータを読み込むためのトグル	parm_set 設定対象パラメタ = 値
ソケットクローズ指定	クライアント側のソケットを閉じる	parm_set 設定対象パラメタ = 値
終了	可視化サブシステムの終了	quit

4.2.2 入力データの区別

ユーザインターフェイス部はデバッガサーバから受け取ったデータを対応する GUI で指定した可視化テンプレートに記述されている入力モジュールと接続されているソケットにデータを書き込む。そのため、ユーザは各デバッガサーバに対応する GUI でそのアプリ

ケーションに対応する入力モジュールを利用するように記述されている可視化テンプレートを選択する。

データ受け渡し用のソケットは、環境ファイルに記述されたポート番号に対応するデバッガサーバのソケット ID を加えたものをポート番号として 4.2.2 に示した手順で作成される。このようにして各アプリケーションに対応するデバッガサーバごとに異なる入力モジュールと接続されたソケットにデータを書き込むことで、可視化サブシステムでデータを区別して比較あるいは組み合わせての表示を行うことができる.。

2 つのアプリケーションのデータの差分を表示するモジュールネットワークの例を図 4.4 に示す

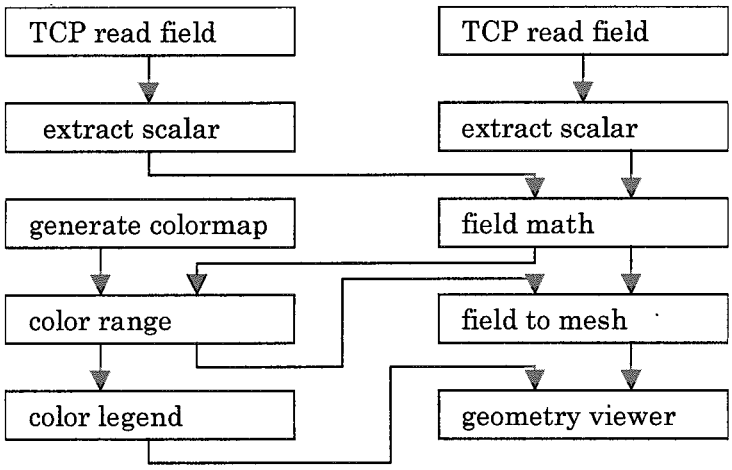

図 4.4 2 つのデータの差分を表示するモジュールネットワーク

5 実行例

2 つのアプリケーションの差分を表示するテンプレートを用いてサンプルプログラムを可視化した例を示す。サンプルとして用いたのは 2 次元の圧縮性流体解析のアプリケーションで翼回りの流体解析の結果である。このプログラムを IBM SP と SGI ONYX で実行 1000 ステップまで計算した結果の差分を表示した。メッシュサイズは、150x50 で、表示したデータは密度の値(単精度実数)の双方のマシンでの計算結果の差である。

5.1 可視化テンプレート

2 つのアプリケーションの差分を表示するためには、同じ CLI ファイルの違う入力モジュールを使う二つの可視化テンプレートを用いることになる。ここでは、2d_diff_0、2d_diff_1 という名称で登録してある 2 つのテンプレートを利用した。この二つのテンプレートの違いは、入力モジュールとして TCP read field.user.0 を用いるか TCP read field.user.1 を用いるかのみである。この 2 つのテンプレートは$CLIDIR にある mesh2d_diff.cli という CLI ファイルを利用している。使用しているモジュール・ネットワークは図 4.4 に示したものであり、その詳細を以下に示す。

(1) 2 つの TCP read field モジュールで得たデータを extract scalar モジュールにより 1 つのスカラーだけを持つフィールドデータとする。
(2) この 2 つのフィールドデータを field math モジュールに渡し、差分を計算する。
(3) この値を field to mesh モジュールと color range モジュールに渡し、メッシュ上の分布を示すジオメトリデータとし、カラーバーとともに geometry viewer モジュールで表示する。

5.2 差分表示のための操作手順

主画面の Communication プルダウンメニューからデバッガサーバとの接続(Connect dbserver)を選択すると図 5.1 に示すデバッガサーバの追加を指定するウインドウが開く。

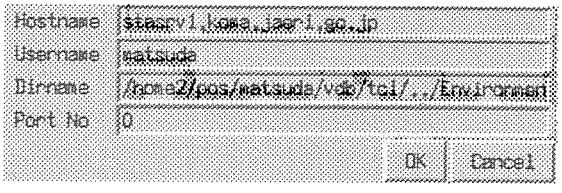

図 5.1 テンプレート F2dmesh に対する変数指定メニュー

このウインドウで、接続するデバッガサーバを動作させるホスト名、ユーザ名、ディレクトリ名、デバッガサーバとユーザインターフェイス部の接続ポート番号を指定する。ポート番号が 0 または無指定の場合には自動的に空いているポート番号が設定される。こうして 2 つ目のデバッガサーバを指定すると、そのデバッガサーバと接続した主画面が新

たに表示される。この2つの主画面でそれぞれ

(1) デバッガの指定
(2) デバッグ対象アプリケーションの指定
(3) 可視化テンプレートの指定
(4) 取得データの指定
(5) データ取得位置の指定
(6) データ送信位置の指定
(7) 実行の指定

を行う。各指定の詳細はデータ可視化機能を持つ並列プログラムデバッグツール：vdebug —vdebug利用手引書—[4]を参照していただきたい。

ここに示す実行例では、IBM SPならびにSGI ONYXともデバッガにはdbxを指定し、アプリケーションにはCFD2D/sw2dを指定した。そして、AVS表示を選択し、可視化テンプレートとしてIBM SPのほうに2d_diff_0、SGI ONYXのほうに2d_diff_1を指定し、取得データとしてxmesh、ymesh、qqの各配列を指定した。取得データの指定の細かい内容は表5.1のとおり。

表 5.1 取得データの指定内容

ラベル	リソース	型	次元	変数名/値	配列範囲
x	program	float	2	xmesh	0:150,1:50
y	program	float	2	ymesh	0:150,1:50
nx	typein	int	0	151	
ny	typein	int	0	50	
field	enum	char	0	irregular	
phy0	program	float	3	qq	0:150,1:50,1:1

データの取得位置および送信位置は図5.2に示すこのアプリケーションの57行目に設定した。

```
 1          program gcfd2d
                                  ～一部省略～
20    c-------------------------------
21    c
22          include 'igas2d.h'
23    c
24          real   qq(ns1:nf1,ns2:nf2,ndim2), qqbar(ns1:nf1,ns2:nf2,ndim2)
25          real   qql(ns1:nf1,ns2:nf2,ndim2),qqr(ns1:nf1,ns2:nf2,ndim2)
26          real   dflux(ns1:nf1,ns2:nf2,ndim2)
27    c
                                  ～一部省略～
44    c
45    c                 1.4 Set up initial condition
46          call setint(qq)
47    c
48    c                 1.5 Output initial data if necessary
49          nseq = 0
50    c*      call output(qq)
51    c*      nseq = nseq + 1
52    c
53    c---------------------------------------------------------------------
54    c                 2. Main Calculation
55    c
56          do 9999 nstep=1,maxstp
57            write(6,*) nstep
58    c
59    c
60    c                 2.1 Calculate time step
61            call timstp2(qq,deltat)
62            time = time + deltat
                                  ～一部省略～
151         stop
152         end
```

図 5.2 データ取得位置ならびに送信位置を指定するファイルの抜粋

これにより図 5.3 に示す 2 つのアプリケーションの差分の表示が行える。

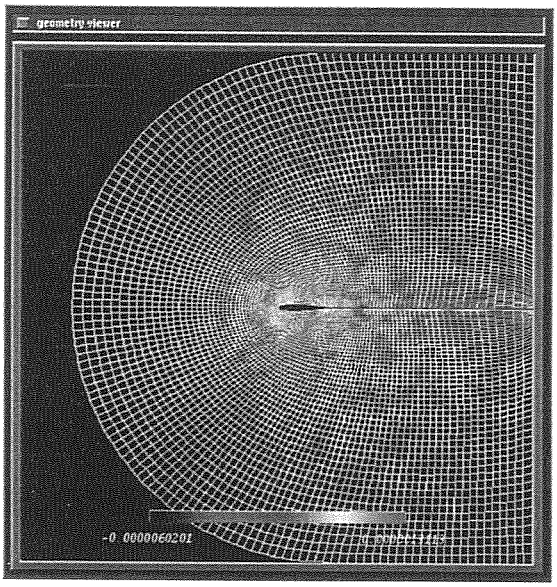

図 5.3 IBM SP と SGI ONYX による流体解析結果の差分表示

This is a blank page.

また、ループの内側のデータ取得位置やデータ送信位置とは別の位置、例えば先のソースの61行目当たりにブレークポイントを設定すれば、それぞれのマシンで1ステップずつ実行した結果を比較していくこともできる。このように同期を取って比較した1ステップ目と2ステップ目の結果を図5.4～7に示す。なお、ループの先頭でデータを取得しているので1ステップ目の値は初期値である。

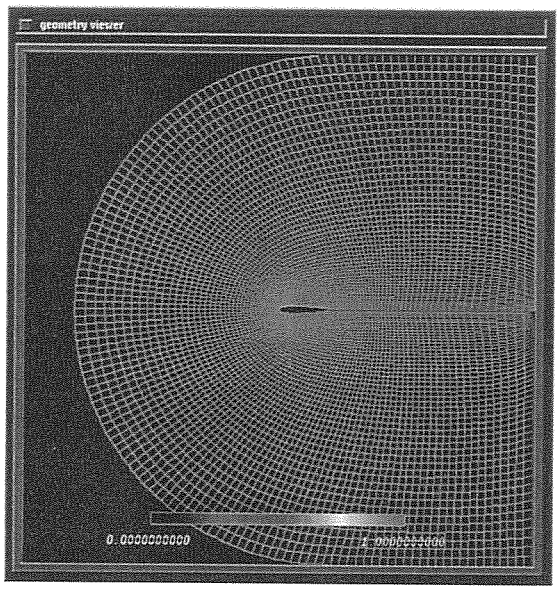

図 5.4 SP だけ1ステップ実行した結果

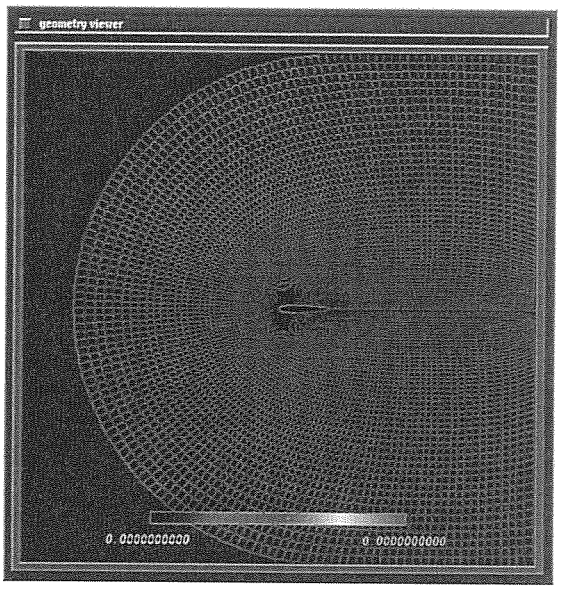

図 5.5 SP と ONYX を1ステップ実行した結果

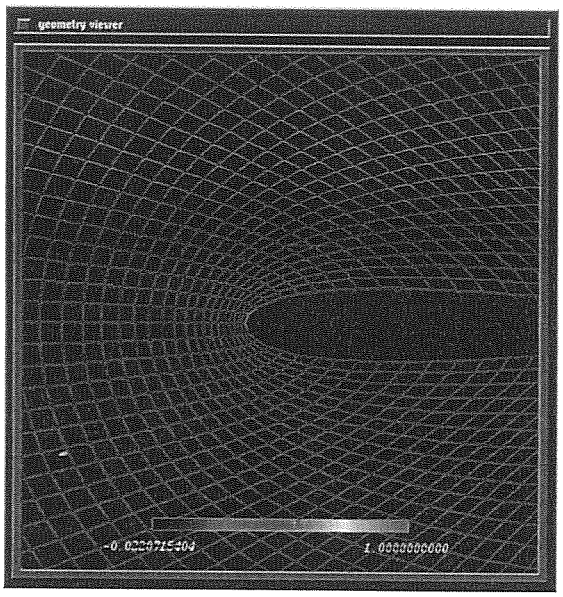

図 5.6 SP2 ステップ、ONYX1 ステップの結果

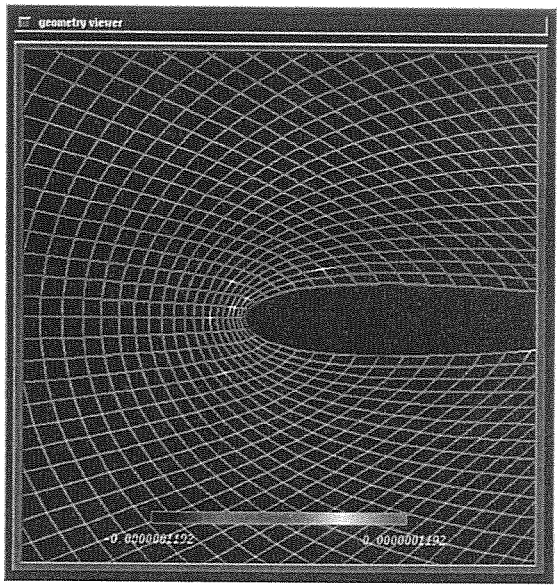

図 5.7 SP と ONYX を2ステップ実行した結果

図5.4はSPの1ステップ目の状態であり、すなわち初期値である。ほぼ全領域で1で右下の出口部分で0という分布になっている。これは、データの定義位置の問題で実際に

This is a blank page.

はグリッドの中心で定義されている密度の値をグリッド上の値として扱ったために起こった問題と思われる。図 5.5 は SP の 1 ステップ目から ONYX の 1 ステップ目の値を引いた結果であり、全領域で 0 になっている。すなわち、どちらのマシンでもすべての領域で同じ値であることが確認できる。図 5.6 は SP だけを次のステップに進めた結果であり、翼の先端部分で若干色が変わっている。カラーバーの値の範囲は-0.22~1.0 であり、1.0 の値を示しているのは図には含まれていないがやはり右下の領域である。図 5.7 は ONYX ももう 1 ステップ進め、どちらも 2 ステップ目に入ったところでの結果である。カラーバーの値は 10^{-7} のオーダーであり,ほぼ等しい値となっていることがわかる。しかしながら、翼の周りで若干赤あるいは青く表示されている部分がある。これは、単精度実数での数値計算の誤差と思われる。

このように、2 つのアプリケーションをじっくり比較しながら計算を進めていくこともできる。今回示した例では、2 つのマシンでの比較をしたわけだが、同じマシンでも違うモデルを利用した場合、パラメタを変更した場合、あるいは並列化をした場合などのさまざまなケースでこの機能を利用することができる。

6 おわりに

可視化デバッガでは大量のデータをデバッガを通してアプリケーションから取得して可視化を行う。それゆえ可視化デバッガの操作性は、データが実用的な速度で取得できるかにかかっている。今回の改良により、テストに用いた程度のサイズのデータであれば簡易なトラッキングシステムとしても十分実用に耐え得るようになったと言えるだろう。また、複数のプログラムを比較したり、組み合わせた可視化表示ができるように複数のデバッガサーバをコントロールできるようにした。これによりプログラムの他機種への移植や並列化、新しいモデルの組み込みなどいろいろな局面でvdebugの利用価値が高まった。

本報告書は、可視化デバッガvdebugの改良に関する開発経過をまとめたものであり、改良項目の設計と追加機能の使用方法が述べてある。本報告書を通じて多くの研究者に可視化デバッガvdebugを利用していただければ幸いである。

なお、可視化デバッガは計算科学技術推進センターCCSEのホームページ[26]において公開されている。研究成果の公開のページを参照していただきたい。

謝　辞

本報告書をまとめるにあたり、日本原子力研究所計算科学技術推進センターの秋元センター長には、貴重な助言をいただいた。ここに深謝いたします。また、同センターの相川次長、平山グループリーダーには、細部に渡り貴重な助言をいただいた。同じく、ここに深謝いたします。

参考文献

[1] 武宮博, 他 : 並列科学計算のための可視化解析システムの開発,計算工学講演会論文集, Vol. 2, No. 1, 1997

[2] 松田勝之, 他 : 可視化デバッガ・システム、vdebug 利用手引き,JAERI-Data/Code 98-036,1998

[3] 松田勝之, 他 : データ可視化機能を持つ並列プログラムデバッグツール:vdebug —vdebug 利用手引書—,JAERI-Data/Code 2000-005,2000

[4] 松田勝之, 他 : データ可視化機能を持つ並列プログラムデバッグツール:vdebug —vdebug プログラム設計書—,JAERI-Data/Code 2000-005,2000

[5] Etnus LLC : http://www.etnus.com/totalview/index.html

[6] 日本電気株式会社 : http://www.leaf.comp.nec.co.jp/lang/psuite/index.html

[7] Sun Microsystems, Inc : the Command-Line Utilities manual

[8] Richard M. Stallman and Roland H. Pesch : Using GDB: A Guide to the GNU

SourceLevel Debugger, 1991

[9] Silicon Graphics, Inc : Compiling Debugging and Performance Tuning

[10] Compaq Computer Corp. : Programer's Guide,
http://www.unix.digital.com/faqs/publications/i18n/japan/V40F/V40F/PROG_GD/TITLE.HTM

[11] International Business Machines Corporation : IBM AIX Version 4 General Programing Concepts: Writing and Debugging Programs

[12] International Business Machines Corporation : IBM Parallel Environment for AIX Operation and Use, Volume 2, Part 1 Debugging and Visualizing, SC28-1980-02

[13] 日本電気株式会社 : DBX 利用の手引き,G1AF19

[14] 日本電気株式会社 : PDBX 利用の手引き,G1AF20

[15] Cray Research Inc. : Introducing the Cray TotalView Debugger

[16] Compaq Computer Corp. : http://www5.compaq.com/products/software/ladebug/

[17] 株式会社日立製作所 : 並列デバッガ ndb 使用の手引,6A20-3-404

[18] 株式会社ケイ・ジー・ティ : AVS ユーザーズ・ガイド

[19] 株式会社ケイ・ジー・ティ : AVS モジュール・リファレンス

[20] 株式会社ケイ・ジー・ティ : AVS/Express ユーザーズ・ガイド

[21] 株式会社ケイ・ジー・ティ : AVS/Express モジュール・リファレンス

[22] 株式会社ケイ・ジー・ティ : AVS/Express デベロッパーズ・ガイド

[23] International Business Machines Corporation : IBM Visualization Data Exprlorer QuickStart Guide, Version 3 Release 1 Modification 4, SC34-3262-02

[24] International Business Machines Corporation : IBM Visualization Data Exprlorer User's Guide, Version 3 Release 1 Modification 4, SC38-0496-06

[25] International Business Machines Corporation : IBM Visualization Data Exprlorer Programmer's Reference, Version 3 Release 1 Modification 4, SC38-0497-06

[26] 日本原子力研究所計算科学技術推進センター : http://guide.tokai.jaeri.go.jp/ccse/index.html

This is a blank page.

国際単位系（SI）と換算表

表1　SI基本単位および補助単位

量	名称	記号
長さ	メートル	m
質量	キログラム	kg
時間	秒	s
電流	アンペア	A
熱力学温度	ケルビン	K
物質量	モル	mol
光度	カンデラ	cd
平面角	ラジアン	rad
立体角	ステラジアン	sr

表2　SIと併用される単位

名称	記号
分，時，日	min, h, d
度，分，秒	°, ′, ″
リットル	l, L
トン	t
電子ボルト	eV
原子質量単位	u

1 eV=1.60218×10^{-19}J

1 u=1.66054×10^{-27}kg

表3　固有の名称をもつSI組立単位

量	名称	記号	他のSI単位による表現
周波数	ヘルツ	Hz	s^{-1}
力	ニュートン	N	$m\cdot kg/s^2$
圧力，応力	パスカル	Pa	N/m^2
エネルギー,仕事,熱量	ジュール	J	N·m
工率，放射束	ワット	W	J/s
電気量，電荷	クーロン	C	A·s
電位，電圧，起電力	ボルト	V	W/A
静電容量	ファラド	F	C/V
電気抵抗	オーム	Ω	V/A
コンダクタンス	ジーメンス	S	A/V
磁束	ウェーバ	Wb	V·s
磁束密度	テスラ	T	Wb/m^2
インダクタンス	ヘンリー	H	Wb/A
セルシウス温度	セルシウス度	℃	
光束	ルーメン	lm	cd·sr
照度	ルクス	lx	lm/m^2
放射能	ベクレル	Bq	s^{-1}
吸収線量	グレイ	Gy	J/kg
線量等量	シーベルト	Sv	J/kg

表4　SIと共に暫定的に維持される単位

名称	記号
オングストローム	Å
バーン	b
バール	bar
ガル	Gal
キュリー	Ci
レントゲン	R
ラド	rad
レム	rem

1 Å=0.1nm=10^{-10}m

1 b=100fm^2=10^{-28}m^2

1 bar=0.1MPa=10^{5}Pa

1 Gal=1cm/s^2=10^{-2}m/s^2

1 Ci=3.7×10^{10}Bq

1 R=2.58×10^{-4}C/kg

1 rad=1cGy=10^{-2}Gy

1 rem=1cSv=10^{-2}Sv

表5　SI接頭語

倍数	接頭語	記号
10^{18}	エクサ	E
10^{15}	ペタ	P
10^{12}	テラ	T
10^{9}	ギガ	G
10^{6}	メガ	M
10^{3}	キロ	k
10^{2}	ヘクト	h
10^{1}	デカ	da
10^{-1}	デシ	d
10^{-2}	センチ	c
10^{-3}	ミリ	m
10^{-6}	マイクロ	μ
10^{-9}	ナノ	n
10^{-12}	ピコ	p
10^{-15}	フェムト	f
10^{-18}	アト	a

(注)

1．表1－5は「国際単位系」第5版，国際度量衡局 1985年刊行による。ただし，1 eVおよび1 uの値はCODATAの1986年推奨値によった。

2．表4には海里，ノット，アール，ヘクタールも含まれているが日常の単位なのでここでは省略した。

3．barは，JISでは流体の圧力を表わす場合に限り表2のカテゴリーに分類されている。

4．EC閣僚理事会指令ではbar，barnおよび「血圧の単位」mmHgを表2のカテゴリーに入れている。

換　算　表

力	N(=10^{5}dyn)	kgf	lbf
	1	0.101972	0.224809
	9.80665	1	2.20462
	4.44822	0.453592	1

粘　度　1 Pa·s(N·s/m^2)=10 P(ポアズ)(g/(cm·s))

動粘度　1m^2/s=10^{4}St(ストークス)(cm^2/s)

圧力	MPa(=10bar)	kgf/cm^2	atm	mmHg(Torr)	lbf/in^2(psi)
	1	10.1972	9.86923	7.50062×10^3	145.038
	0.0980665	1	0.967841	735.559	14.2233
	0.101325	1.03323	1	760	14.6959
	1.33322×10^{-4}	1.35951×10^{-3}	1.31579×10^{-3}	1	1.93368×10^{-2}
	6.89476×10^{-3}	7.03070×10^{-2}	6.80460×10^{-2}	51.7149	1

エネルギー・仕事・熱量	J(=10^7 erg)	kgf·m	kW·h	cal(計量法)	Btu	ft·lbf	eV
	1	0.101972	2.77778×10^{-7}	0.238889	9.47813×10^{-4}	0.737562	6.24150×10^{18}
	9.80665	1	2.72407×10^{-6}	2.34270	9.29487×10^{-3}	7.23301	6.12082×10^{19}
	3.6×10^6	3.67098×10^5	1	8.59999×10^5	3412.13	2.65522×10^6	2.24694×10^{25}
	4.18605	0.426858	1.16279×10^{-6}	1	3.96759×10^{-3}	3.08747	2.61272×10^{19}
	1055.06	107.586	2.93072×10^{-4}	252.042	1	778.172	6.58515×10^{21}
	1.35582	0.138255	3.76616×10^{-7}	0.323890	1.28506×10^{-3}	1	8.46233×10^{18}
	1.60218×10^{-19}	1.63377×10^{-20}	4.45050×10^{-26}	3.82743×10^{-20}	1.51857×10^{-22}	1.18171×10^{-19}	1

1 cal= 4.18605J（計量法）

= 4.184J（熱化学）

= 4.1855J（15℃）

= 4.1868J（国際蒸気表）

仕事率　1 PS(仏馬力)

= 75 kgf·m/s

= 735.499W

放射能	Bq	Ci
	1	2.70270×10^{-11}
	3.7×10^{10}	1

吸収線量	Gy	rad
	1	100
	0.01	1

照射線量	C/kg	R
	1	3876
	2.58×10^{-4}	1

線量当量	Sv	rem
	1	100
	0.01	1

(86年12月26日現在)

可視化デバッガの改良
―データ取得機能の高速化と複数のプログラムの比較・組み合わせ表示機能の開発―