並列プログラムデバッグのための可視化ツール

小柳 滋[†] 久保田 和人[††] 川倉 康嗣[†]

並列プログラミングを困難なものとしている要因のひとつに，並列デバッギングの困難性があげられる．従来の並列デバッガは通信の動作状況をユーザに示すことに重点が置かれていたが，これだけでは不十分であり，プログラムの動作状況をユーザが理解しやすい形に加工して表示することが有効である．本稿では，並列プログラムの挙動を簡易に可視化することのできるツールの提案，試作および評価を行う．提案するツールは，既存の逐次デバッガを通じてプログラム中の変数の値を取得し，その値をもとにプログラムの挙動の可視化を行う．このため，可視化に際しソースプログラムの変更は必要ない．また，どのように表示するかについてユーザの意図を反映するため，可視化のためのテンプレートを動的に選択できる構成をとる．さらに，並列プログラムの停止位置や停止条件の設定を柔軟に行えるという特徴を持つ．本可視化ツールをワークステーションクラスタ上の仮想計算機（PVM）上に実装し，いくつかの例題を用いて効果を確認した．

Parallel Visualizing Debugger

SHIGERU OYANAGI,[†] KAZUTO KUBOTA[††] and YASUSHI KAWAKURA[†]

One major reason for the difficulty of parallel programming is that parallel programs are hard to debug. Conventional parallel debuggers have focused on the analysis and representation of communication patterns. It is useful to visualize the behavior of the parallel program based on the user's view. A parallel visualizing debugger is proposed and developed. The debugger consists of two parts, a debugger and a visualizer. The debugger provides flexible stop conditions in a parallel program. The visualizer obtains variable values through the debugger without modifying the source program. Then the visualizer displays the behavior of the program based on a template which the user can select dynamically. The parallel visualizing debugger has been implemented on a workstation cluster and has been applied successfully to several parallel programs.

1. まえがき

並列プログラミングを困難なものとしている要因のひとつに，並列デバッギングの困難性があげられる．並列プログラムでは，並列プログラムを構成する個々の部分プログラムが相互に通信を行いながら処理が進められていくため，複雑な挙動となる．また，非決定的なプログラムにおいてはバグの再現性が問題となり，デバッグをさらに困難にしている．これまで研究，開発されてきた並列デバッガは，通信の動作状況をユーザに示すことに重点が置かれていた．メッセージパッシングライブラリPVM[1]上のトレースツールであるXab[2]やExpress[3]のEtool，並列計算機Paragon上のプログラミングツールParAide[4]，ParaGraph[5]等のツールでは，通信の状況をトレースファイルに格納し可視化することができる．また，並列プログラムの実行を再現する再演法の研究としてはInstant Replay法[6]が有名であり，再演法を用いたデバッガの実現法も研究されている[7]．

このようなツールは並列プログラムをデバッグするうえで必要不可欠なものであるが，これらのツールだけでは発見が困難なバグが存在する．たとえば，通信命令の対応に誤りがある並列プログラムが正常終了した場合は，ユーザが通信命令の対応に誤りがあること自体を認識することは難しい．また，通信部分を調べたとしても，膨大な通信ログの中から対応に誤りがある部分を発見することは難しい．このようなバグの原因を特定するには，プログラムの動作状況をユーザに的確に提供することが有効である．しかしながら，並列プログラムの挙動は複雑であり取り扱うデータ量も多いため，単に情報を羅列してユーザに提供してもユーザが状況を把握することは難しい．ここで重要

† 株式会社東芝研究開発センター
TOSHIBA Research & Development Center
†† 新情報処理開発機構
Real World Computing Partnership

なのは，通信状況だけでなくプログラムの動作状況をユーザが理解しやすい形に加工して見せるということであり，その最も効果的な方法はプログラムの挙動をグラフなり図形なりに描画してユーザに見せるという，いわゆる可視化を行うことである．

本稿では，並列プログラムのデバッグのための可視化技術に焦点をあてて議論を行う．可視化技術としては，前述の通信の状況を可視化するアプローチの他にプログラム中の変数の値を可視化するアプローチもある．たとえば並列計算機 MasPar のプログラミング環境である MPPE[8] や CM5 の Prism[9]，AP1000[10] のパラレルデバッガは，プログラム中の変数の値を，その値に応じた色で表示する機能を持つ．また，PVM のプログラミングツールである HeNCE[11] では，ユーザが記述したプログラムのフローを表すグラフ上で，プログラムの動作を見ることができる．さらに，Voyeur View[12] では，プログラムの挙動をユーザの希望する形態で可視化する研究を行っている．

我々は，並列プログラムのデバッグのための可視化として，以下のような要件が重要であると考えている．

(1) デバッグのための可視化であるので，ソースプログラムの書き換えは避けること．
(2) どのように表示するかについてのユーザの意図を反映できること．
(3) 可視化された画面を表示専用とするのでなく，さらなるデバッグ情報を得るためのユーザの操作を可能とすること．

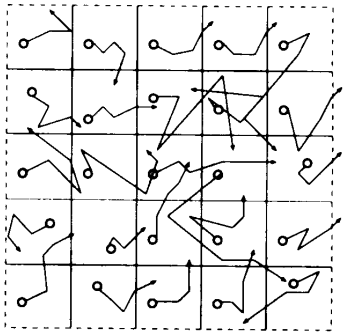
以上のような考え方に基づいて，本稿ではプログラムの挙動をユーザの視点から簡易に可視化することのできるツールを提案，試作し，その評価を行う．提案する可視化ツールは，既存のデバッガを通じてプログラム中の変数の値を取得し，その値をもとにプログラムの挙動の可視化を行うものである．したがって，可視化関数を埋め込む等のソースプログラムの変更は必要ない．また，既存のデバッガ上に構築されているため，停止位置や停止条件の設定が柔軟に行えるという特徴を持つ．

本稿では，2 章で本可視化ツールの概要および実現する機能について述べ，3 章では，本システムの構成およびインプリメンテーションについて述べる．4 章では本システムの操作事例について述べ，5 章では考察を行う．6 章ではまとめと今後の課題について述べる．

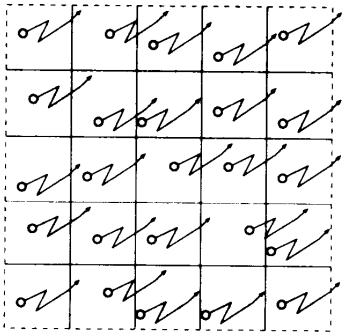
2. 可視化ツールの概要

2.1 ツールの位置付けとプログラミングモデル

並列プログラムに固有のバグとして，部分プログラム間のメッセージの不具合によるものがあげられる．たとえば，メッセージパッシング型のプログラムにおいて，send 命令と receive 命令の対応を誤ったために生じるデッドロックは，並列プログラムにおける典型的なバグである．このため，メッセージパターンをユーザに表示することでデバッグを支援する技術が数多く研究されてきた．しかしながら，メッセージパターンのみに着目しても発見が困難なバグが存在する．図 1 に例を示す．ユーザは二次元平面上にランダムに動く粒子の様子をシミュレーションしたいものとする．二次元平面を領域分割し，小領域ごとにプロセッサを割り当て，小領域内の粒子は，個々のプロセッサでシミュレーションされる．プログラムが正しく動作した場合には，図 1 (a) に示したように粒子はランダムに動作する．しかしながら，各プロセッサで同系列の乱数が発生するバグが生じると，図 1 (b) に示すように，各領域での粒子の動作はまったく同じものになってしまう．このようなバグは，プログラム内部のデータの挙動を可視化して表示することが有効な手段となる．本可視化ツールは，このようなメッセージパターンの表示解析だけでは発見が困難なバグのデバッグ支援を目的とする．



(a) ランダムな粒子のシミュレーション



(b) 同系列の乱数発生によるバグ

図 1 可視化が有効な例
Fig. 1 An example for visual debugging.

今回作成した可視化ツールは，SPMD 型のプログラムを対象とした．これは，現在の科学技術計算等で

は広くSPMD型のプログラムが使われていることによる．

2.2 実現する機能

本可視化ツールは，以下に示す機能の実現を目指して開発した．

(1) デバッガと可視化ツールの連動
デバッガを通じてプログラム中の変数の値を取得し，デバッガとは別の可視化ツールにより表示を行う．すなわち，デバッガは並列プログラム中にブレークポイントを設定することにより実行を停止させ，プログラム中の変数の値を取得し，これを可視化ツールに渡してプログラムの実行を継続させる．これにより，可視化に際してソースファイルにグラフィックのための関数等を埋め込む必要がなくなる．

(2) ユーザの意図の反映
上記可視化ツールにおいて，可視化のためのテンプレートを複数用意しておき，ユーザが適したテンプレートを選択できるようにする．たとえば，ユーザが2次元平面上に飛散する粒子の様子を表示したい場合には二次元の表示のテンプレートを，三次元の場合には三次元のテンプレートをといった具合に，ユーザの意図した観点からの表示を可能にする．また，ユーザの意図する表示方法が用意されているテンプレートで実現できない場合には，簡単な記述で所望のテンプレートが得られるような環境を提供する．これにより，ユーザの視点からの可視化を可能にする．

(3) 可視化ツールからのフィードバック
可視化ツールによって表示された画面からユーザは何らかの情報を読み取ることになる．読み取った情報からユーザが次に何をすべきかが判断できればよいが，表示された情報が不十分で，ユーザはさらに詳しい情報を必要とする場合がある．そこで，グラフィック画面を操作することでユーザに必要な情報が得られれば有益である．たとえば，

- グラフィック画面上にプロットされた点をクリックすると，その点を計算したプロセッサが表示される機能
- プログラムが複数のブレークポイントで停止している場合に，グラフック画面上にプロットされた点をクリックすると，その点を計算したプロセッサの停止位置を表示する機能

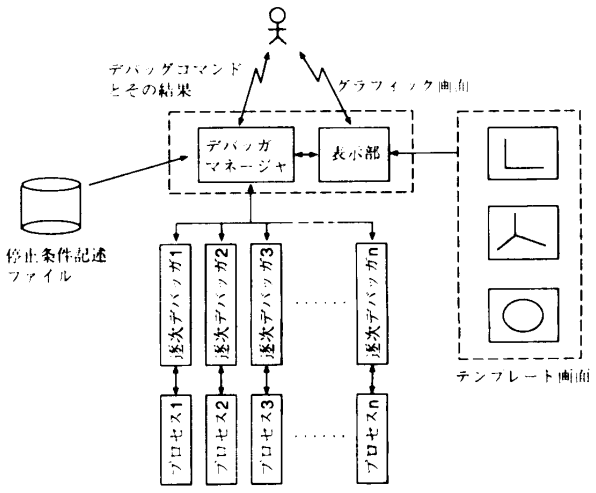


図2 システム構成
Fig. 2 System organization.

等の機能を持たせれば，ユーザがデバッグを進めていく過程で，状況の把握および不具合のある箇所の探索が効率的になるだろう．ここで重要なのは，可視化された画面をユーザが情報を得るための終着点とするのではなく，得られた画面をさらに深い情報を得るための中間点とし，デバッガに対して何らかのフィードバックをかけたり，ソースプログラムに対するポインタを指し示したりという，可視化ツールからのアクションを可能とすることである．

3. インプリメンテーション

本章ではシステム構成およびインプリメント方法について示す．

3.1 構成と動作

本可視化ツールは，仮想並列計算機PVM (Parallel Virtual Machine)[1] 用に記述されたプログラムを対象とした．システム全体の構成を図 2 に示す．個々のプロセスは，それぞれ逐次デバッガで制御される．逐次デバッガは対応するユーザプログラムにブレークポイントを設定し，実行を停止させてプログラム中の変数の値を取得する．それぞれの逐次デバッガに対する入出力はデバッガマネージャと呼ばれるツールで統合される．デバッガマネージャはまた並列プログラムを柔軟に停止させるため，後述する条件停止の機能を実現する．この停止条件を記述するために条件記述ファイルが用意される．さらに，デバッガマネージャによって取得された情報は表示部に渡され可視化が行われる．可視化の際の画面記述としてテンプレートが用意される．

ユーザのデバッグコマンドはデバッガマネージャを

```
for { set i 0 } { $i < $PENUM } { incr i } {
    if { VAR[$i:y] > 100 } {
        STOP
    }
}
```

図3 停止条件の記述
Fig. 3 Description of stop condition.

通じて個々の逐次デバッガへと送られ，その結果がユーザへと返される．通常はユーザはすべての逐次デバッガとやりとりを行うが，やりとりを行う逐次デバッガを指定する機能も設ける．なお，ツール全体として，グラフィカルなユーザインタフェースを用意する．

3.2 条件停止の実現

前述のように，本システムではデバッガにより並列プログラムの実行を停止させて可視化に必要な情報を取得する．そのためには，並列プログラムの停止条件を柔軟に設定できることが必要となる．ここでは，並列プログラムを構成する部分プログラム間で特定の条件が満たされたとき，プログラムを停止させるという条件停止機能の実現法について述べる．ユーザが複数の部分プログラムにまたがる条件でプログラムを停止させたい場合には，条件記述ファイルに条件をプログラム形式で記述する．このプログラムは，Tcl/Tk[13]で記述する．今，ユーザがすべての部分プログラム（これらは，すべて同一のものとする）の同一行に対してブレークポイントを設定し，プログラムを実行させたとする．すべてのプログラムがブレークポイントに到達すると，停止条件記述ファイルに記述してあるプログラムがデバッガマネージャ内のTcl/Tkインタープリタによって解釈実行される．今，**図3**に示すようなプログラムが記述されていたとする．ここで，**$PENUM**は部分プログラムの数を表す変数，**VAR[$i:y]**は，i番目のプログラムにおける変数yの値とする．**STOP**はすべてのプログラムをブレークポイントで停止させる．この条件記述では，すべての部分プログラム間に設定されたブレークポイント間で疑似的にバリア同期がとられ，同期がとられたときの各部分プログラム中の変数yについて，**y > 100**のものが1つでもあればすべてのプログラムは停止し，そうでない場合は継続実行される．

このように停止条件をプログラム形式で記述することにより，柔軟な停止条件の記述が可能となる．

3.3 可視化部分の実装

可視化を行うにあたって，**show at**コマンドと**probe**コマンドという2つのユーザコマンドを設けた．これらのコマンドはデバッガマネージャで解釈される．**probe**コマンドは，ソースファイルの行番号と変数名を引数として受け取る．この行をプローブポイントと呼び，変数名をプローブ変数と呼ぶ．プログラムがプローブポイントに到達したとき，プローブ変数の内容がデバッガマネージャによって取得される．**show at**コマンドは，ソースファイルの行番号位置を引数として受け取る．この行をショウポイントと呼ぶ．プログラムがショウポイントに到達したときユーザに対する表示が行われる．ユーザがプローブポイントとショウポイントを設定するごとに，デバッガマネージャはその行に対して逐次デバッガを通じてブレークポイントを設定する．**show at**コマンドや**probe**コマンドに前述の停止条件の記述を組み合わせることも可能である．プローブポイントとショウポイントの行番号はデバッガマネージャ内で保持される．ユーザがプログラムを実行した後，デバッガマネージャは，個々の部分プログラムがいずれかのブレークポイントに到達するか，プログラムが終了するのかを待つ．ある部分プログラムが到達したブレークポイントがプローブポイントだった場合は，その部分プログラムを制御する逐次デバッガに**print**コマンドを送って変数の値を取得する．ショウポイントだった場合は，プローブポイントで取得している変数を表示部に送る．

表示部はユーザが選択したテンプレートの内容に基づいて可視化を行う．ユーザはテンプレートを選択時に，テンプレート変数というテンプレート内部の変数とプログラム中の変数の対応を記述する（**図4**）．テンプレート変数とは，仮に，ユーザが3次元の点をプロットするテンプレートを選択し，そのテンプレートがX，Y，Z軸を持っていた場合に，X，Y，Zをテンプレート変数と呼ぶ．ユーザがプログラム中のp，q，rという変数を表示したい場合には，$p \rightarrow X$，$q \rightarrow Y$，$r \rightarrow Z$というプログラム変数とテンプレート変数の対応を記述することになる．ユーザの希望するテンプレートがない場合は，ユーザが自分でテンプレートを作成することもできる．現在は，ユーザがTcl/Tkでテンプレートプログラムを記述しデバッガマネージャに登録することで，ユーザテンプレートの追加が行える．

3.4 フィードバックの実現

前述のように，可視化ツールにより表示された画面を操作することにより，さらなる情報を得るための機能をフィードバックと呼ぶ．ここでは，フィードバックの一例として，表示画面からプロセッサマップへフィードバックをかける方法について説明する．オンラインデバッガを通じて取得されたプローブ変数は，そのプローブ変数が取得されたプロセッサ番号とともにデ

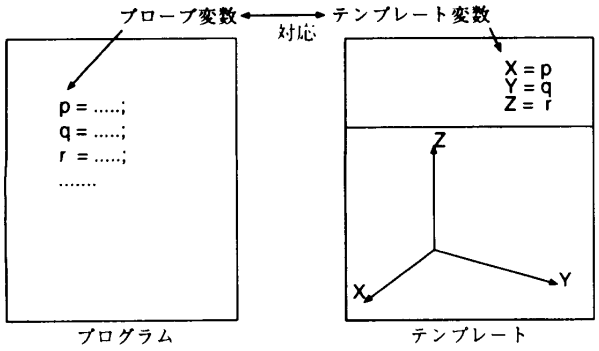
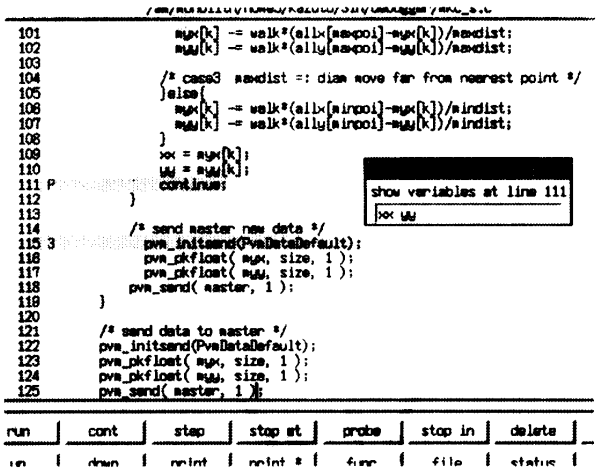


図 4 変数対応の記述
Fig. 4 Correspondence between variables.

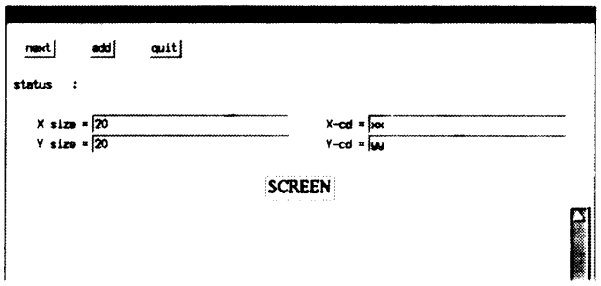
バッガマネージャ内のプローブ変数格納領域に表形式で格納されている．また，デバッガマネージャ内にはプローブ変数とテンプレート変数の対応表が作成される．ユーザが，表示画面上の物体をクリックすると，その物体を描画するのに用いられたプローブ変数が，対応表を参照することにより求められる．プローブ変数が求まると，デバッガマネージャ内のプローブ変数格納領域の表を検索することで，対応するプロセッサの番号を特定することができ，プロセッサマップ上でその番号のプロセッサを表示することができる．このようにテンプレートによる画面記述方式を採用したことにより，任意の画面からのフィードバックが容易に実現できる．

4. 操 作 例

本章では，試作した並列デバッガで並列プログラムの可視化を行った例を示す．例題として円整列問題[14]を解くプログラムを用いる．円整列問題とは，「無秩序な初期状態から個体同士が互いに情報を交換しつつ，中央集権的な制御機構なしに自律分散的に円周という秩序的な形を形成するアルゴリズム」であり，ミルウォーキ大の鈴木によって考案されたものである．このアルゴリズムを，C 言語と PVM を用いて並列プログラム化した．それぞれの個体が運動し，円へと収束していく様子を可視化する．作成したプログラム中で，各個体の座標を保持している部分は図 **5** (a) の 109，110 行目の xx，yy という変数である．111 行目にプローブポイントを設定する．111 行目をクリックすることでウインドウが開き，そこで取得したい変数を記述する．次に，115 行目にショウポイントを設定する．テンプレートとして，2 次元のグラフを選択する．テンプレート変数は，X 軸を表す X，Y 軸を表す Y からなる．図 5 は，選択された 2 次元のテンプレートに対する変数対応の記述例である．表の項目の，



(a) プローブポイントの設定



(b) 変数対応の記述

図 5 可視化のための設定
Fig. 5 Setup for visualization.

$X-cd=$，$Y-cd=$ に，それぞれ xx，yy を記入することで変数の対応をとる．プログラムを実行させるとプログラムはショウポイントで停止し，停止時点の個体の位置が可視化される．図 **6** は，収束ループを 16 回繰り返した後の絵である．白抜きの点が現在の個体の位置，黒い点が各粒子の動いた軌跡である．現時点で，粒子がほぼ円周上に並んでいるのが分かる．同じプログラムを，3 次元グラフのテンプレートを用いて表示した絵が図 **7** である．縦軸がループの回数を表しており，ループの回数を重ねるごとに，散らばっていた個体が円周上に集まっていく様子が観察される．このように，本可視化ツールを用いればプログラムの挙動をオンラインで把握することができるようになる．

他のプログラムを可視化した例を以下に示す．

- ガウスの消去法
  ガウスの消去法を用いて連立方程式の解を求めるプログラムの動作状況を可視化している（図 **8**）．平面に描かれた格子が行列を表しており，黒色の濃い要素（15 個ある）がすでに計算が終わった要素，灰色の要素（5 個ある）が現在計算中の要素である．この計算は 5 個のプロセスで並列に行っており，5 カ所の要素が並列に計算されている様

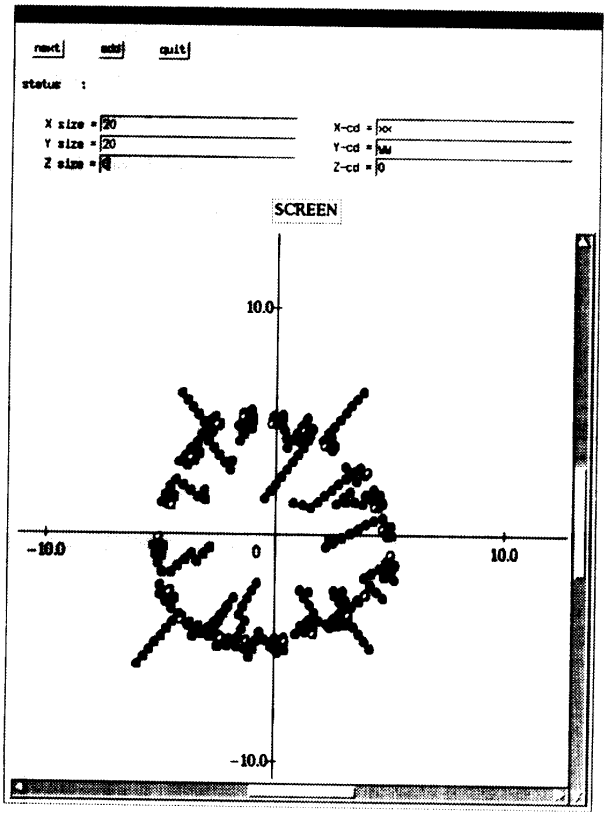


図6 2次元テンプレートによる可視化
Fig. 6 Visualization with two-dimensional template.

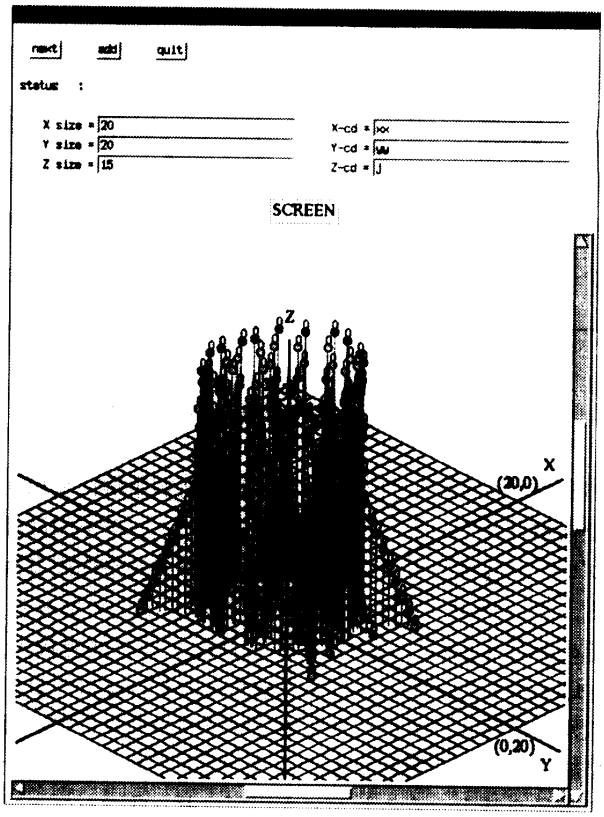


図7 3次元表示による可視化
Fig. 7 Visualization with three-dimensional template.

子が分かる.

- 微分方程式の求解
  差分法を用いて, $d^2x/dy^2 + y = -1$ という式の解を求めるプログラムの動作状況を表した絵が**図 9** である. 図中の上側の境界の点の集合が, 現在の解の曲線を表している. 処理が進んでいくと, この点の集合が上方向に移動していき, 最終的にはサイン曲線に収束するべきである. しかしながら, このプログラムにはバグが入っているためサインには収束しない. 図 9 の曲線を見ると左右対称ではないので, あらかじめプログラムが対称な曲線に収束すると分かっているプログラマには, プログラムの不具合が認識できる. 通常のデバッガで, 変数の値を出力してもこのようなバグは認識しにくいが, このように絵に出せばバグの存在が一目瞭然となる.

5. 考　　察

本章では, 試作した並列デバッガに関する問題点を指摘し, その改善方法について考察する.

5.1 並列プログラムの停止と同期

並列プログラムの挙動を可視化する際には, どの時点でプログラムを停止させて変数の値を取得するのかということが問題となる. SPMD 型のプログラムにおいて, すべての部分プログラムを同一箇所で停止させ, その状況を見ることは容易である. しかしながら, プログラムによっては個々のプログラムのバラバラの地点におけるデータを集計して表示したい場合がある. ここでは, 複数のプログラムが異なるブレークポイントで停止する場合の問題点について述べる.

複数のプロセッサ上で, **図 10** に示す 2 重ループ構造のプログラムが動作しているものとする. 内側のループは収束計算を行うループで, 収束ループを回る回数はプロセッサごとに異なる. ユーザは変数 e の値を見たいものとする. e の値は, 02 行目の if 文の条件によって 03 行目で計算される場合と 05 行目で計算される場合がある. いま, ユーザは**表 1** に示すように, i の値が同一である場合の収束ループ中の e の値を集めたい場合があると考えられる. この表は, i が 0 のときプロセッサ 0 から 4 は収束ループをそれぞれ 1, 4, 3, 3, 2 回通過していることを示しており, ユーザが集めたいのは 1 回目は 0.1, 5.2, 6.4, 9.2, 1.2 という 5 個の e の値, 2 回目は 2.5, 2.4, 3.8, 0.8 という 4 個の e の値である. いま, 03 行目と 05 行目の 2 カ所だけにブレークポイントを設定してプログラムを実行し, すべてのプログラムがブレークポイントに到

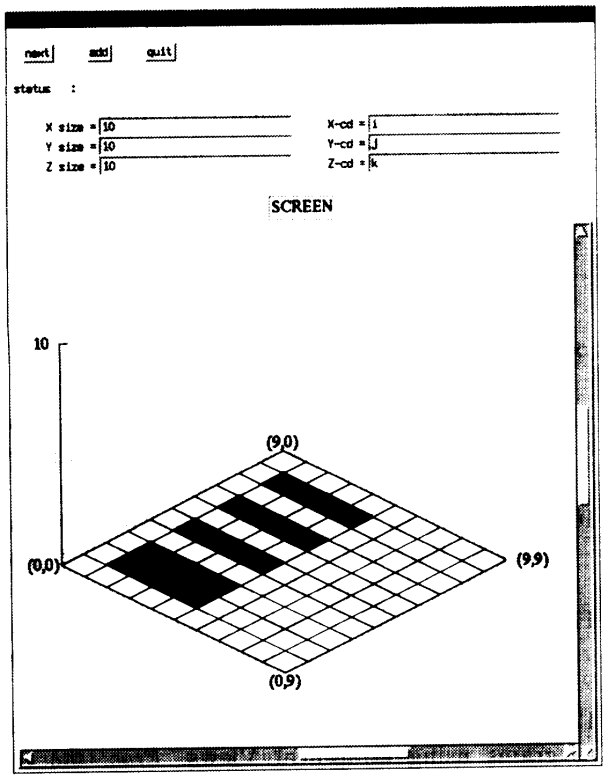


図8 ガウスの消去法の可視化
Fig. 8 Visualization of gaussian elimination.

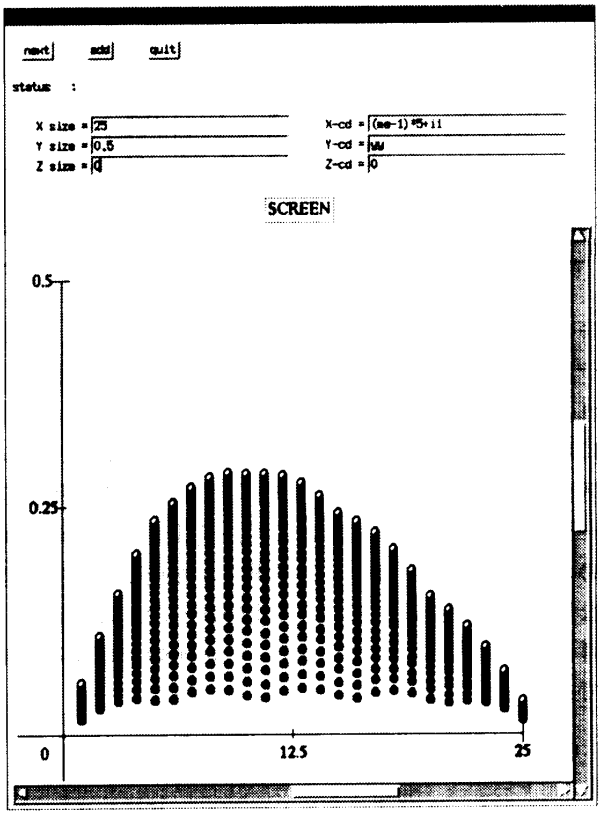


図9 微分方程式の求解
Fig. 9 Visualization Of Differential equation.

達したあとですべてのプロセッサから e の値を集め，再びすべてのプロセッサに継続命令を出したとする．この方法だと，内側の収束ループが早く終わったプロセッサは，i 値が更新された後，再び収束ループを回り 03 か 05 行目のブレークポイントで停止するので，異なる i のときの e の値が集められてしまいユーザの要求には答えられない．

この要求に応えるためには 07 行目にバリア同期を設定する必要がある．本可視化ツールの目標であるようなソースプログラムの書き換えをすることなくバリア同期を設定するためには，07 行目にブレークポイントを設定し，部分プログラムがブレークポイントに到達したときに疑似的なバリア同期をとる機構が並列デバッガの内部に必要となる．

5.2 高 速 化

本システムのインプリメンテーションでは，以下の点において速度的な問題がある．

(1) デバッガにより変数値を取得するため，逐次デバッガとユーザプログラム間ではコンテクストスイッチが，また，デバッガマネージャと逐次デバッガ間ではプロセス間通信が，それぞれ頻発する．

(2) 表示部に Tcl/Tk を用いているため，表示処理に時間を要する．

```
00 for( i=0; i<100; i++ ){ /*外側ループ*/
01   while( e > 0.1 ){     /*収束ループ*/
02     if( y > 0 ){
03       e = ...;
04     }else{
05       e = ...;
06     }
07   }
08 }
```

図10 複数箇所でのプログラムの停止
Fig. 10 Sample program with multiple breakpoints.

後者については表示処理専用のプログラムを開発すれば解決する課題であり，検討の対象外とし，以下では前者について検討する．

可視化したい変数値の取得にデバッガを用いる利点はインタラクティブに操作できることである．並列プログラムのデバッグの初期段階においてはインタラクティブ性は重要であるが，大規模並列プログラムのデバッグにおいて基本的な制御の流れ等に関する初期デバッグが終了し，プログラムを連続的に動作させて全体的な動き等をデバッグする時点では，インタラク

表1 サンプルプログラムの挙動
Table 1 Behavior of the sample program.

i=		0				1			2		3
プロセッサ	0	0.1	-	-	-	0.1	-	-	0.1	-	.....
	1	5.2	2.5	1.4	0.9	6.4	0.9	0.2	1.4	0.7	.....
	2	6.4	2.4	0.4	-	1.2	0.8	-	0.2	-	.....
	3	9.2	3.8	0.2	-	5.2	2.4	0.4	3.7	0.2	.....
	4	1.2	0.8	-	-	2.5	0.3	-	3.8	0.4	.....

ティブ性よりも実行性能が重要となる．そこで，初期デバッグ段階でデバッガにより可視化のために設定されたコマンドに対応する機能をソースプログラムの中に埋め込み，コンパイルする手法が考えられる．本システムのインプリメンテーションにおいてこの手法を一部取り入れて実験したところ，変数値の取得のためのオーバヘッドがほとんど無視できる程度にまで大幅な性能向上が見込まれることが分かった．このような考え方を導入した本格的なシステム構築が今後の課題である．

6. あとがき

本稿では，並列プログラムのデバッグのためにプログラムの挙動を可視化する機能を持つツールの提案，試作および評価を行った．本可視化ツールは，従来の並列デバッガがプログラム間の通信命令を追いかけることに重点がおかれていたのに対し，ユーザの視点から並列プログラムの挙動を可視化することができる．本可視化ツールは，従来の並列デバッガを置き換えるものではなく，従来のデバッガでは不足していた機能を補うものである．従来の通信命令の把握に重点を置くデバッガと本可視化ツールを組み合わせることにより，さらなる並列プログラムのデバッグの効率化が期待できる．

今後は，本可視化ツールの適用範囲をSPMD型のプログラムからMIMD型へと広げていく予定である．

参考文献

1) Geist, G.A., et al.: PVM 3 User's Guide and Reference Manual (1994).
2) Beguelin, A.L.: Xab: A Tool for Monitoring PVM Programs, *1993 Workshop on Heterogeneous Processing* (1993).
3) ParaSoft Corp.: Express Fortran User's Guide Version 3.0 (1990).
4) Bemmerl, T.: Programming Tools for Massively Parallel Supercomputers, *Environments and Tools for Parallel Scientific Computing*, pp.125–136 (1993).
5) Heath, M.T., et al.: Visualizing the Performance of Parallel Programs, *IEEE Software*, Vol.8, pp.29–39 (1991).
6) LeBlanc, T.J. and Mellor-Crummey, J.: Debugging Parallel Programs with Instant Replay, *IEEE Trans. Comput.*, Vol.C-36, pp.471–482 (1987).
7) 三栄，高橋：適応的再演型ロック命令を用いた並列プログラムデバッガの実現，*JSPP'94*, pp.241–248 (1994).
8) MasPar Computer Corporation: Data-Parallel Programming Guide (1991).
9) Thinking Machines Corporation. The Connection Machine CM-5 Technical Summary (1992).
10) Horie, T., et al.: AP1000 Software Environment for Parallel Programming, *FUJITSU Sci. Tech. J.*, Vol.29, pp.25–31 (1993).
11) Geist, A., et al.: Visualization and Debugging in a Heterogeneous Environment, *IEEE Computer*, Vol.26, pp.88–95 (1993).
12) Bailey, M.L., et al.: Debugging Parallel Programs using Graphical Views, *Proc. the International Conference on Parallel Processing*, pp.46–49 (1988).
13) Ousterhout: *Tcl and the Tk Toolkit*, Addison Wesley (1994).
14) 吉田：超並列協調計算モデルに関する研究，超並列原理に基づく情報処理基本体系，第2回シンポジウム予稿集，pp.27–29 (1993).

(平成7年9月4日受付)
(平成8年4月12日採録)

小柳 滋（正会員）

昭和24年生．昭和47年京都大学工学部数理工学科卒業．昭和52年同大学院博士課程修了．同年（株）東芝入社．現在，同社研究開発センター情報・通信システム研究所研究主幹，並列計算機アーキテクチャ，および並列処理・分散処理応用に関心を持つ．工学博士，電子情報通信学会，IEEE，ACM各会員

久保田和人（正会員）

昭和 39 年生．昭和 63 年早稲田大学理工学部電子通信学科卒業平成 2 年同大学院修士課程修了．平成 5 年同大学院博士後期課程修了．同年（株）東芝入社．平成 7 年 10 月より新情報処理開発機構に出向中．並列計算機のプログラミング環境に関する研究に従事．工学博士．電子情報通信学会，ソフトウェア科学会各会員．

川倉 康嗣（正会員）

昭和 63 年京都大学工学部情報工学科卒業．平成 2 年同大学院修士課程修了．同年（株）東芝入社．現在，同社研究開発センター情報・通信システム研究所勤務．並列計算機の利用技術，プログラミング環境の研究・開発に従事．電子情報通信学会会員．