変数オリエンティッドなデータ依存関係モデルの提案

笹倉 万里子[†1] 城 和貴[†2]
國枝 義敏[†3] 荒木 啓二郎[†4]

プログラムを特徴づける性質の一つとしてデータ依存関係がある．これまで，データ依存関係は，ソースプログラムの文の間の関係としてコンパイラの内部表現で表されていた．しかし近年，ユーザによるプログラムの理解，最適化や並列化の観点から，ユーザによるデータ依存関係の理解が重要性を増している．そこで本論文では，ユーザへのデータ依存関係の提示を主目的として，変数を中心としてデータ依存関係を記述するモデルを提案し，それを用いたデータ依存関係の視覚化法について述べる．また，提案するデータ依存関係の表現法と従来の表現法との対応について述べる．

A Variable-oriented Data Dependence Model

Mariko Sasakura,[†1] Kazuki Joe ,[†2] Yoshitoshi Kunieda [†3]
and Keijiro Araki[†4]

Data dependence is one of the most important information which characterizes a program. Usually, in compiles, data dependence is expressed as relation between sentences of program code. Users do not see the expression. However, recently, it is more and more important for users to understand data dependence for optimization and/or parallelization. Therefore, we propose a new model for data dependence which is expressed in view of variables, and we visualize data dependence based on the model. We also mention the relation between usual model and our model.

1. はじめに

データ依存関係は，あるプログラムにおいて発生するあるデータへの参照の間に存在する関係で，そのプログラムを特徴づける性質の一つである．プログラムの最適化，デバッグ，あるいは逐次プログラムを並列プログラムに変換する並列化といったプログラム変換において，変換の前後でデータ依存関係が変化するか否かは，その変換が正当であるかどうかを判断する上で最も重要な情報の一つである．

これまで，プログラムの最適化や並列化といったプログラムをより効率良く実行するためのプログラム変換は，コンパイラによって自動的に行われ，ユーザは一旦プログラムを書いたらそのプログラムの性質について知る必要はなかった．しかし，プログラムが大規模かつ複雑になってくるにつれ，プログラムのデバッグやメンテナンスのために，すでに書かれたプログラムの性質をユーザが理解することが重要となってきている．また，最適化や並列化に関しても，完全に自動的に行うよりもユーザの知識を利用する方がより効率の良いプログラムに変換することができるため，プログラムの性質をユーザが理解することが重要である．分散環境におけるプログラムの分散実行での最も重要な問題の一つであるデータ分割においても，HPF（High Performance Fortran）のようにユーザに分割の仕方を指定させる場合が多い．

このように，プログラムの性質をユーザが理解していることは重要であるが，ユーザがプログラムの性質をソースコードから直接読みとるのは一般に非常に困難である．確かに，プログラムの性質はソースコードの中に表現されているものではあるが，明示的な形でソースコードに現れているわけではない．そこで，ユーザのプログラムの理解を支援するシステムが必要になる．

例えば，ParaScope[5] やParassist[6] は，ユーザの知

†1 岡山大学工学部
Faculty of Engineering, Okayama University
†2 和歌山大学システム工学部
Faculty of Systems Engineering, Wakayama University
†3 和歌山大学システム情報学センター
Center for Information Science, Wakayama University
†4 九州大学大学院システム情報科学研究科
Graduate School of Information Science and Electrical Engineering, Kyushu University

識を利用してプログラムをより並列化しようとするシステムで，これらのシステムでは，プログラムの性質をユーザが理解するのを助けるために，コンパイラがプログラムを解析した結果をユーザに提示している．しかし，

- コンパイラ内部で使用することを目的としてプログラムを解析した結果をそのままユーザに提示している．
- 解析の表示がテキスト中心である．

という理由から，ユーザにとってわかりやすく情報を提示しているとはいい難い．実際5)では，ユーザからの要望として，解析を充実することと共に，ユーザインタフェースの充実が挙げられている．

そこで，本論文で我々はユーザが最も理解すべきプログラムの性質の一つであるデータ依存関係について，それをユーザが理解しやすいように表現することを考える．すなわち，変数，配列といったデータを中心としてデータ依存関係記述するモデルを提案し，それを用いたデータ依存関係の視覚化法について述べる．また，提案するデータ依存関係の表現法と従来の表現法との対応について述べる．

これまで，データ依存関係は，コンパイラの内部表現として，文を基本単位としたグラフとして表現されるのが普通であった．すなわち，文を一つのノードとし，同じデータに対して参照を行うノード同士の間を方向付の枝で結んでデータ依存関係を表す．コンパイラ内部では最適化の最小単位を文とするため，このように文という決まった単位でデータ依存関係を表現することで，コンパイラ内部での最適化の操作が簡単になっていた．この表現の方法では二つのノードで具体的にどの変数が共通に参照されているかがグラフには明示されない．そのため，依存の原因となっている変数を明確にするため，グラフの枝に変数名をラベルとして添えるという手段をとられる場合もある．

従来，データ依存関係の表現で変数が重要視されていなかったのは，文を最小単位としてプログラム変換を行っていたからである．しかし，ユーザがプログラムの性質を理解しようとする時には，データ依存関係の本質，すなわち，変数を中心としてデータ依存関係を表現するのが妥当であると考えられる．また，ユーザは文という固定された単位だけではなく，必要に応じて，ループや関数といった大きな単位，あるいは，文よりも細かな命令レベルなど，さまざまな粒度でのデータ依存関係を見ることでプログラムのデータ依存関係を把握しやすくなると考えられる．本論文で提案するデータ依存関係表現は，変数，配列を中心としたモデルであり，変数を中心と考えたことにより，文の単位だけでなく，さまざまな粒度のデータ依存関係を同一の形式で表現できることが特徴である．

まず，2節では，従来のデータ依存関係の表現について簡単にまとめる．3節で，提案する変数オリエンティッドなデータ依存関係表現のモデルを説明する．4節で提案するモデルと従来の表現とを比較し，考察を行う．そして5節では，提案するデータ依存関係モデルに基づくデータ依存関係の視覚化方法について簡単に述べ，6節で視覚化されたデータ依存関係の例を示す．最後に7節でまとめを行う．

2. 従来のデータ依存関係の表現

データ依存関係の解析法および表現については，1), 4), 9), 10)などがあるが，どれもコンパイラ内部で用いることを前提とし，文を単位としてデータ依存関係を表現しているという点で同じである．

ここでは，その一つとしてBanerjeeの定義[3]を紹介する．Banerjeeはデータ依存とその依存の種類について次のように明確に定義している．

代入文Sの実行S'と代入文Tの実行T'がある時，

(1) S'とT'で同じデータを参照していて，その参照のうち少なくとも一つが書き込み参照であり，
(2) S'の方がT'より先に実行され，
(3) そのデータに対してS'とT'の間で書き込みが行われない時，

文Sから文Tに依存があるといい，その種類は，

- S'で書き込み，T'で読み込みの時フロー依存（flow dependence），
- S'で読み込み，T'で書き込みの時逆依存（anti dependence），
- S'で書き込み，T'で書き込みの時出力依存（output dependence），

である．

この定義は，特に自動並列化のためのコンパイラで一般的に用いられている．

3. 変数オリエンティッドなデータ依存関係表現

本論文で提案するデータ依存関係表現は，変数に対するアクセスを定義し，そのアクセス間の関係としてデータ依存関係を定義する．ここで新たにデータ依存関係を定義する目的は，変数の値がどのアクセスからどのアクセスまで保存されなければならないか（これはすなわち変数の生存期間[1]である），また，変数の値がどのアクセスからどのアクセスの間では保存される必要がないの

かを明示することである．

アクセスを明確に定義するための準備として，変数の参照を定義する．変数の参照はマシン語における変数へのアクセスに対応する．

定義 3.1 (変数の参照) 変数 v に対する参照を $R(v,t,k)$ の三つ組で表現する．ここで，v は変数または配列のある要素（これらをまとめて本論文では変数もしくはデータと呼ぶ），t は時刻，k は参照の種類で，k の値は読み込み（R），書き込み（W）のいずれかである．ここで，t は十分細かく，同じ時刻ではあるデータに対しては一回の参照しか行われないとする．

アクセスはこの変数の参照の集合として定義される．アクセスの順序を示す概念時刻を次のように定義する．

定義 3.2 (概念時刻) 概念時刻 t はプログラム順序を示し，$t_i < t_j$ とは，t_i で実行される命令が t_j で実行される命令より先に起こることを示す．

概念時刻には，様々な粒度の単位を考えることができる．例えば次のようなものが考えられる．

- 変数の読み込み，書き込みをそれぞれ一単位とするもの．すなわち，変数の参照の時刻を概念時刻として用いる．これを参照タイムと呼ぶ．
- 概念時刻 t の間にはある変数に対して書き込みがたかだか一度しかなく，かつ，書き込みがある場合にはその書き込みが t の間に生じる参照の最後であると保証されているもの．Fortran の代入文がこれに該当するため，我々はこれをステートメントアクセスタイムと呼ぶ．
- ループの場合に，ループのイタレーション[3] を単位としたもの．これをイタレーションアクセスタイムと呼ぶ．例えば完全二重ループがあり外側のループ回数が n，内側が m，その二重ループの中に文が k 文あったとすると，ステートメントアクセスタイムではこのループ全体のアクセスを $n \times m \times k$ 個の時刻で表現するが，内側のループに着目したイタレーションアクセスタイム，すなわち，各イタレーションを一つの単位とした概念時刻では $n \times m$ 個の時刻で表現する．また，外側のループに着目したイタレーションアクセスタイム，すなわち，外側のループのイタレーションを単位とした概念時刻では n 個の時刻で表現する．

この概念時刻を用いてアクセスを次のように定義する．

定義 3.3 (アクセス) 変数 v に対するアクセス $A(v,t,k)$ は，概念時刻 t 中に起こる変数 v へのすべての参照の集合である．ここで，k はアクセスの種類で，k の値は読み込み（R），書き込み（W），読み書き（R&W）のいずれかであり，以下のように決まる．

読み込み（R） 概念時刻 t において変数 v に対して行われる参照がすべて読み込みである．

書き込み（W） 概念時刻 t において変数 v に対して行われる参照のうち，もっとも最初に行われる参照が書き込みである．

読み書き（R&W） 概念時刻 t において変数 v に対して行われる参照の少なくとも一つが書き込みであり，かつ，もっとも最初に行われる参照が読み込みである．

例えば，概念時刻として参照タイムを用いた場合には，各アクセスでは変数に対する参照が必ず一つであるので，アクセスの種類は読み込み，書き込みのいずれかとなり，読み書きである場合は存在しない．ステートメントアクセスタイムを用いた場合には，3 種類すべてがあり得るが，ステートメントタイムの定義から，アクセスの種類が書き込みであるとは t 中にはその変数は読み込まれないということを意味し，アクセスの種類が読み書きであるとは t 中にその変数が一度以上読み込まれた後書き込まれるということを意味する．イタレーションアクセスタイムを用いた場合には，アクセスの種類が読み込みであるとは，そのイタレーションの間にはその変数に対しては読み込み参照しか行われないということを示し，アクセスの種類が書き込みであるとは，そのイタレーションの間に最初に書き込みが行われた後，0 回以上の書き込みまたは読み込み参照が行われることを意味する．また，アクセスの種類が読み書きであるとは，そのイタレーションの間に，最初に読み込み参照が行われた後，1 回以上の書き込みアクセスが存在することを意味する．

我々は，このアクセスをもとに変数オリエンティッドなデータ依存を次のように定義する．

定義 3.4 (変数オリエンティッドなデータ依存) あるデータ d に対し，アクセス $A_i(d,t_i,k_i)$, $A_j(d,t_j,k_j)$ (ただし $t_i < t_j$) がある時，

(1) 次の条件が満たされる時，d の t_i から t_j にかけて W-R 依存があるといい，$WR(d,t_i,t_j)$ と表す．
- (a) $(k_i = (W \vee R\&W))$ かつ
- (b) $(k_j = (R \vee R\&W))$ かつ
- (c) $t_i < t_k < t_j, k_k = (W \vee R\&W)$ となる $A_k(d,t_k,k_k)$ が存在しない．

(2) 次の条件が満たされる時，d の t_i から t_j にかけて R-W 依存があるといい，$RW(d,t_i,t_j)$ と表す．
- (a) $k_j = W$ で

(b) $t_i < t_k < t_j$ となる $A_k(d, t_k, k_k)$ が存在しない.

この定義ではアクセス間の関係として依存関係を定義している. その結果, さまざまな粒度において, 変数を中心としたデータ依存関係が容易に表現できる. また, 概念時刻の条件さえ満たしていれば, 異なる粒度間のアクセスでも依存関係を定義できる. そして, 最長のW-R 依存は変数の生存期間と一致する.

イタレーションアクセスタイムのように, ステートメントアクセスタイムより粗い粒度の概念時刻を用いてアクセスを表現した時, アクセスの種類が読み書きであるとは, そのアクセスの内部にデータ依存関係が存在することを意味している. 従って, 読み書きアクセスの部分をステートメントアクセスタイムのようにより細かな粒度の概念時刻を用いて表現すれば, データ依存関係をさらに詳細に調べることができる.

概念時刻を参照タイム, ステートメントアクセスタイムとした場合には, 特別にW-W 依存を以下のように定義することができる.

定義 3.5 (W-W 依存) あるデータ d に対し, アクセス $A_i(d, t_i, k_i)$, $A_j(d, t_j, k_j)$ (ただし $t_i < t_j$) があり, $t_i < t_k < t_j$ となる $A_k(d, t_k, k_k)$ が存在せず, $k_i = (W \vee R\&W)$ かつ $k_j = W$ の時, t_i から t_j にかけてW-W 依存があるという.

W-W 依存は, 変数に書き込まれたが一度も読まれることのなかった値の存在を明示している.

4. 変数オリエンティッドなデータ依存関係表現と従来のデータ依存関係

本節では, 2節で述べたBanerjee のフロー依存, 逆依存, 出力依存と3節で定義したデータ依存関係との対応を示す. まず, Banerjee のデータ依存関係の表現の基本となっている文ごとのデータ依存は, 我々のモデルで, 次のように簡単に表すことができる.

文 S と文 T の間の依存の集合は, すべてのデータ d に関して,

$$\{WR(d, S', T') | WR(d, S', T') \in DEP\} \cup$$
$$\{RW(d, S', T') | RW(d, S', T') \in DEP\}$$

ただし, ここで, DEP はすべての依存の集合で, S', T' はそれぞれ文 S, T の実行である.

フロー依存, 逆依存, 出力依存とW-R 依存, R-W 依存の関係は次のようになる.

命題 4.1 フロー依存の集合とW-R 依存の集合は一致する.

証明 4.1 これは定義より自明である.

命題 4.2 R-W 依存は, 逆依存か出力依存のどちらかまたは両方である.

証明 4.2 定義よりR-W 依存であるのは, 連続する二つのアクセス A_i, A_j の種類 k_i, k_j がそれぞれ, R, W の場合, W, W の場合, RW, W の場合である. k_i, k_j が R, W である時, それは逆依存, W, W の時は出力依存, RW, W の時は逆依存と出力依存の両方である. 逆に, k_i, k_j が W, RW 時は出力依存ではあるがR-W 依存ではなく, R, RW の時は逆依存であるがR-W 依存ではない.

概念時刻を参照タイム, ステートメントアクセスタイムとした場合に定義できるW-W 依存と出力依存の関係は以下である.

命題 4.3 W-W 依存の集合は, 出力依存の集合の部分集合である.

証明 4.3 定義よりW-W 依存であれは出力依存である. しかし, 出力依存であってもW-W 依存であるとは限らない. 例えば, 書き込み−読み込み−書き込みの場合, 最初の書き込みから後の書き込みへは出力依存があるが, W-W 依存はない.

図1 に変数オリエンティッドなデータ依存関係表現とBanerjee が定義したデータ依存関係の対応を示す. Banerjee の定義では, コンパイラが最適化・並列化のためにプログラムの文の順序を変更しようとした時, データ依存関係の観点からその変更が許諾できるものかどうかを二つの文の間のデータ依存関係だけから判断することができる. しかし, 我々のR-W 依存やW-W 依存ではもともとのプログラムで隣あったアクセス同士の関係しか示していないために, 二つのアクセスの間の依存関係のみからはそれを判断することは難しい. しかし, 6 節で述べるように, すべてのアクセスを視覚化して見れば判断することは容易である,

一方, 変数の値が保持されなければならない期間, 保持される必要がない期間を明示する, という点では我々のデータ依存関係表現が適している. 我々のデータ依存関係表現では, 変数の値が保持される期間はまさしくW-R 依存で表現されている. 変数の値が保持される必要がない時は, R-W 依存やW-W 依存で表されている. Banerjee が定義した逆依存, 出力依存では, アクセス $A_i(d, t_i, k_i), A_k(d, t_k, k_k), A_j(d, t_j, k_j), t_i < t_k < t_j$ があり $(k_i, k_k, k_j) = (R, R, W)$ の場合にも A_i のアクセスが生じる文と A_j のアクセスが生じる文の間に逆依存があるとする. 出力依存に関しても同じである. そのため, 変数の値が保持される必要がない期間を逆依存, 出力依存の有無のみから判断することはできない.

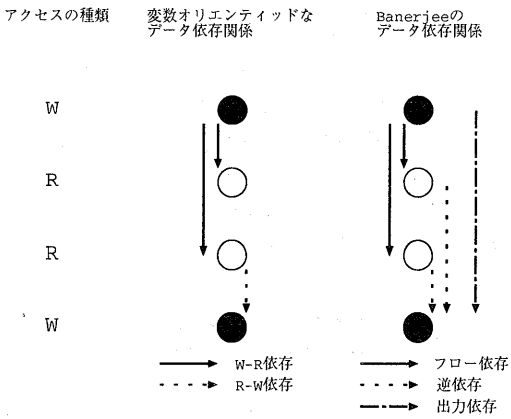


図1 変数オリエンティッドなデータ依存関係と Banerjee のデータ依存関係
Fig. 1 Our variable-oriented Data Dependence Model and Banerjee's data dependence model

表1 キューブの属性と情報の対応
Table 1 The correspondence of properties of a cube and information

キューブ	情報
x,y	アクセスされるデータ
z	アクセスが行われる概念時刻
色	アクセスの種類

5. 変数オリエンティッドなデータ依存関係表現を使ったデータ依存関係の視覚化

5.1 表示要素

この節では，データ依存関係を視覚化する方法について述べる．我々はデータアクセスをキューブで，データ依存関係をポールで表現する．それぞれの変数がどの座標に割り当てられているかを示すために AVD マップを用意する．また，ループにおいては，データ依存関係とループとの関係を明確に示すためにループグリッドを用いてループの区切りを明示する．

5.1.1 キューブ

キューブとは各アクセスごとに作成される立方体である．すなわち，あるアクセスと一対一にキューブを対応させ表示する．各キューブの持つ座標値 (x, y, z) および色は**表1**のようにアクセスの情報と対応付ける．

小さな画面で多くの情報が表示でき，ユーザにとって全体のパターンが直観的にわかるようにするために，我々は，データを x-y 平面上に配置する．配置の仕方はユーザが指定することができる．

z 軸は時間軸とし，概念時刻を対応させる．視覚化システムでは概念時刻としてイタレーションアクセスタイム，ステートメントアクセスタイムを切り替えて表示することができる．必要に応じてステートメントアクセスタイムとイタレーションアクセスタイムを混在させることも可能であるので，多重ループの場合外側のループをステートメントアクセスタイムで，内側のループをイタレーションアクセスタイムで表示するなどの柔軟な指定も可能である．

表2 アクセスの種類と色の対応
Table 2 The correspondence of a kind of access to a color

アクセスの種類	色
R	青
W	赤
R&W	紫

表3 ポールの種類と色の対応
Table 3 The correspondence of a kind of pole to a color

ポールの種類	色
W-R ポール	緑
R-W ポール	黄
W-W ポール	ピンク

アクセスの種類は色彩で区別する．その対応を**表2**に示す．

5.1.2 ポール

データ依存関係をユーザに明示するために，アクセスを示すキューブ間をポール（円柱）でつなぐ．ポールには，W-R ポール，R-W ポール，W-W ポールの3種類があり，それぞれ3節で述べた W-R 依存，R-W 依存，W-W 依存を表現している．それぞれのポールの種類を一目で区別できるように各ポールを異なった色で表示する（**表3**）．

依存の種類の定義から，3種類のポールをすべて表示しても，異なる種類のポールが同じ座標に表示されることはないことがわかるが，3次元表示では，見る角度によってポールが重なって見づらくなる場合があるので，ユーザはポールの種類ごとに表示・非表示を選択することができる．また，6節で例で示すように，それぞれの変数に値が書き込まれた後最初に読み込まれるのがいつかを見ることがユーザには有用であると考えるので，視覚化システムではこれを最も短い W-R 依存のポールと呼び，そのポールだけを表示する機能を備える．

5.1.3 AVD (Array-Variable Disposition) マップ

AVD マップは，データが x-y 平面上にどのように配置されているかを示す平面である．平面上のある点 (x,y) に，そこに配置されているデータの変数名を表示する．

前述したように AVD マップ上へのデータの配置の仕方はユーザが決めることができる．通常は，AVD マップ上の一つの (x,y) 座標に一つの変数，または配列の一

つの要素を配置するが，いくつかの配列要素（例えば，ある行，ある列など）に対して一つの(x,y)座標を対応させることも可能である．また，ユーザは必要に応じてAVDマップそのものの表示・非表示を指定できる．

5.1.4 ループグリッド

ループにおけるデータ依存関係を表示する際には，ループによって生じる繰り返しが，表示されているどの部分にあたるのかを明確にすることが重要だと考えられる．そこで，アクセスとループ構造の関係を明確にするために，指定した概念時刻の位置にx-y平面に平行に半透明な平面を表示する．これをループグリッドと呼ぶことにする．

ユーザは各イタレーションの最初，あるいは特定のイタレーションにループグリッドを表示するように指示することができる．例えば二重ループの外側のループの各イタレーションの始まりに対応させてループグリッドを表示すると，外側ループにより生じるアクセスやデータ依存関係のパターンが把握しやすくなる．また，ユーザは必要に応じてループグリッドの表示・非表示を指定することができる．

5.2 視覚化された図から得られる情報

このような方法でデータ依存関係を視覚化した図から，我々は最適化，並列化，デバッグなどに有用な情報を以下のように読みとることができる．

- W-Rポールによって，変数の生存期間がわかり，意図しない読み込みアクセス，書き込みアクセスの存在がわかり得る．
- W-Wポールが存在したら，そのW-Wポールの始まりの書き込みアクセスで書き込まれた値は使われなかったことがわかる．
- すべてのポールを表示した時に，$z=i$の平面をどのポールも横切らなかったら，$z=i$の前の部分と後の部分はデータ依存関係の観点ではどちらを先に実行しても結果は変わらない．
- すべてのポールを表示した時に，$i<z<j$の間で始まりかつ終るポールが存在しなければ，その区間内のアクセスはデータ依存関係の観点ではどのような順番で実行しても結果は変わらない．
- AVDマップでの変数の配置をメモリの配置と考えることで，データ分散の問題に用いることができる．

6. 例

ここでは，まず，一般的なループの例として7)の例題プログラムに変数を付け加えた**図2**のFortranプログラムに対する従来のデータ依存関係，変数オリエンティッドな依存関係，変数オリエンティッドな依存関係の視覚化の例を示し，変数オリエンティッドな依存関係を視覚化することで，従来の依存関係より直観的にデータ依存関係を把握できることを示す．次にプログラムの並列化に効果がある例としてウェーブフロント型のプログラムのデータ依存関係とそれを並列化したプログラムのデータ依存関係の図を示す．

```
      do 20 i = 3, 10
        do 18 j = 5, 10
          m = a(i, j) - b(i, j)  : S1
          a(i, j) = b(i-3, j-5)  : S2
          b(i, j) = a(i-2, j-4)  : S3
          c(i, j) = m - b(i, j)  : S4
   18   continue
   20 continue
```

図2 プログラム例1
Fig. 2 Sample program 1

6.1 一般的なループの例

6.1.1 従来のデータ依存関係

従来の依存関係の表現では，図2のプログラムに存在する依存は次のように表現される．

(1) S_1からS_1に出力依存
(2) S_1からS_2に逆依存
(3) S_1からS_3に逆依存
(4) S_1からS_4にフロー依存
(5) S_2からS_3にフロー依存
(6) S_3からS_2にフロー依存
(7) S_3からS_4にフロー依存
(8) S_4からS_1に逆依存

1，4，8がmに関する依存，2，5がaに関する依存，3，6，7がbに関する依存である．

6.1.2 変数オリエンティッドなデータ依存関係とそれによる視覚化

まず，データ依存関係の概略を見るために，図2のプログラムの依存関係を内側のループに着目したイタレーションアクセスタイムの変数オリエンティッドなデータ依存関係で表現すると，配列a，配列bに関して読み書きアクセスがあり，変数mに関して書き込みアクセスがあることがわかる．そこで，データ依存関係をより細かく見るためにステートメントアクセスタイムを用いてデータ依存関係を表現したのが，**表4**である．ここで，概念時刻は$S_k(i,j)$と表現されている．これは，イタレーション(i,j)の時の文S_kの実行であることを示している．

これを視覚化したものを**図4**，**図5**に示す．どちら

```
      do 10 i = 2, n-1
        do 20 j = 2, m-1
          a(i, j) = (a(i-1, j) + a(i, j-1)
            + a(i+1, j) + a(i, j+1))/4
20      continue
10    continue
```

図3 プログラム例2
Fig. 3 Sample program 2

も，図の左奥から右前に向けてx軸，左前から右奥に向けてy軸，下から上へ向けてz軸となっている．すなわち，この図では図の下の方のアクセスの方が上の方のアクセスより先に行われる．ループグリッドは外側のループに対応したもののみ表示している．図4の方がW-Rポールを表示したもの，図5がR-Wポールを表示したものである．このサンプルプログラムではW-W依存はなく，W-Wポールは存在しないので，W-Wポールを表示する図は省略する．変数のマッピングは図の下の部分にAVDマップとして表示されているが，おおよそ図の左から，配列a，配列b，変数m，配列cの順である．

この図を見れば，アクセス・データ依存のパターンを直観的に把握できる．

6.2 ウェーブフロント型のループの例

ウェーブフロント型のループとは**図3**のプログラムのように計算がマトリックス上を波のように伝播していくものをいう．ループスキューイング[2]はウェーブフロント型のループを並列化する一つの手法である．このようなループは，通常の並列化コンパイラでは，ユーザがこのループに対してスキューイングを行うよう指示することで並列化できる．

図6は図3のデータ依存関係を視覚化したものである．ここでは，n = 10，m = 10とし，ループグリッドは外側のループに対応するもののみ表示している．概念時刻はステートメントアクセスタイムを用いているが，実際にはこのプログラムはループの本体が代入文一文のみから構成されているので，概念時刻としてステートメントアクセスタイムを用いた場合と内側のループに着目したイタレーションアクセスタイムを用いた場合で，データ依存関係表現は実質的に同じになり，視覚化して得られる図も同一のものとなる．

ウェーブフロント型のループにおけるデータへのアクセスの特徴は，あるデータへの書き込みが行なわれた後，そのデータへの読み込みが，内側のループを実行している間に一度あり，さらに外側のループをまわる時にもう一度あることである．**図7**では，その特徴がよくわかるように最も短いW-R依存のポールだけを表示している．

これらの図からこのループがデータ依存関係により並列化できないことが次のように読みとれる．

- 図6に見られるように，ループグリッドをまたがるポールが存在することから，外側のループに関しては実行順序が変更できず，外側のループは並列実行できない．
- 図7に見られるように，ループグリッドをまたがらないポールが存在することから，ループグリッドの間，すなわち内側のループでの実行の順序が変更できず，内側のループは並列実行できない．

このプログラムにループスキューイングを施して得られたプログラムのデータ依存関係を**図8**に示す．この図を見ると，すべてのデータについて，赤のキューブと二つの青のキューブがループグリッドによって隔てられていることがわかる．これをさらに明確にわかるように最も短いW-Rポールだけを表示させたものが**図9**である．これより，このループグリッドにはさまれている部分，すなわち内側のループが，並列に実行可能であることが容易にわかる．

このように，視覚化された図により，簡単にもとのプログラムのデータ依存関係の特徴を把握して，適用可能な並列化手法を選択することができる．さらに，並列化手法適用後のプログラムのデータ依存関係を見ることで，確かに並列化できることを確認することが可能である．

7. おわりに

本論文では，データ依存関係の新しい表現法として変数オリエンティッドなデータ依存関係を定義した．そして，従来のBanerjeeのデータ依存関係表現がコンパイラの内部表現として文中心であるのに対し，我々の提案した変数オリエンティッドなデータ依存関係が変数の値が保存されなければならない期間，保存される必要がない期間を明示しているという点で，変数中心となっていることを示した．我々が提案したデータ依存関係表現では，概念時刻を用いることによりさまざまな粒度でのデータ依存関係を統一的に扱うことができる．

また，本論文では，変数オリエンティッドなデータ依存関係を用いて表現されたデータ依存関係を容易に視覚化する方法を示し，それによって生成された図の例を提示した．この図によってアクセス・データ依存関係が直観的に把握できるだけでなく，最適化や並列化に重要な情報を得られることを例を用いて示した．

本論文で提案した変数オリエンティッドなデータ依存

表4 変数オリエンティッドなデータ依存関係で表現したサンプルプログラムの依存関係
Table 4 Data dependence of sample program 1 by our variable-oritented data dependence model

$WR(m, S_1(3,5), S_4(3,5)), \ldots, WR(m, S_1(10,10), S_4(10,10))$
$RW(m, S_4(3,5), S_1(3,6)), \ldots, RW(m, S_4(10,9), S_1(10,10))$
$RW(a(3,5), S_1(3,5), S_2(3,5)), WR(a(3,5), S_2(3,5), S_3(5,9))$
$RW(a(3,6), S_1(3,6), S_2(3,6)), WR(a(3,6), S_2(3,6), S_3(5,10))$
...
$RW(b(3,5), S_1(3,5), S_3(3,5)), WR(b(3,5), S_3(3,5), S_4(3,5)), WR(b(3,5), S_3(3,5), S_2(6,10))$
$RW(b(3,6), S_1(3,6), S_3(3,6)), WR(b(3,6), S_3(3,6), S_4(3,6))$
...
$RW(b(10,10), S_1(10,10), S_3(10,10)), WR(b(10,10), S_3(10,10), S_4(10,10))$

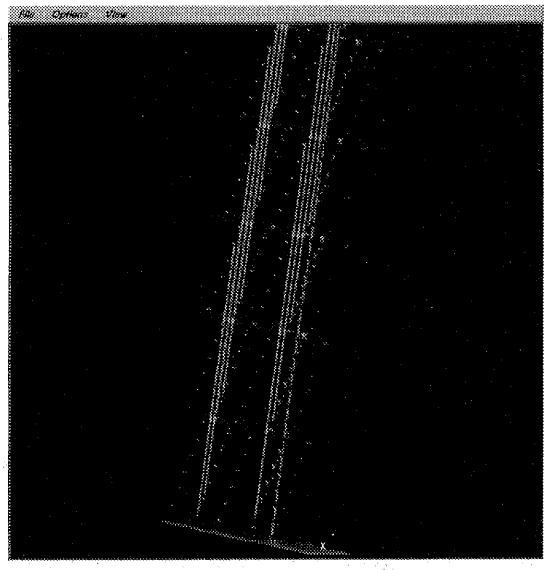


図4 図2の依存関係の視覚化（W-Rポール）
Fig. 4 Visualizaton of data dependence of Figure2(with W-R pole)

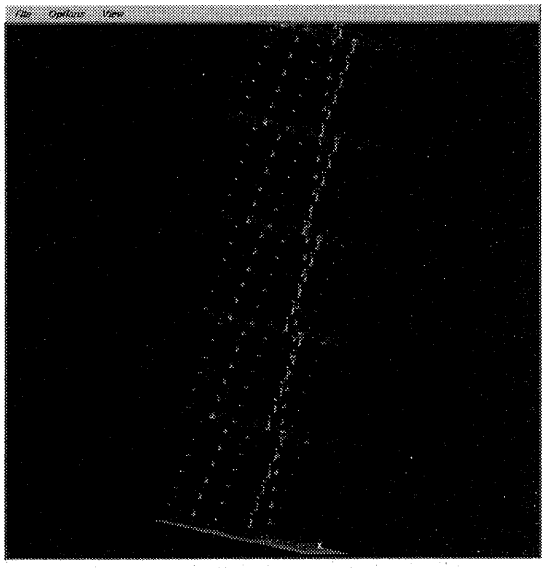


図5 図2の依存関係の視覚化（R-Wポール）
Fig. 5 Visualizaton of data dependence of Figure2(with R-W pole)

関係表現とそれに基づくデータ依存関係の視覚化は，現在並列計算において重要な課題である分散メモリ上へのデータ分割問題を見据えたものでもある．分散メモリ上へのデータ分割においては，変数への参照がいつ生じるかをさまざまな粒度で調べながら，実際にどの変数をどのメモリに配置するかを決定する必要がある．本論文で提案したデータ依存関係表現を基にして，分散メモリ上へのデータ分割問題に関する支援環境を構築することが今後の課題である．

本論文で述べたデータ依存関係の視覚化法は，並列化支援視覚化システム NaraView[8), 11)] に組み込まれている．

参考文献

1) Aho, A., Sethi, R. and Ulman, J.: *Compilers: Principles, Techniques and Tools*, Addison-Wesley (1986).
2) Bacon, D. F., Graham, S. L. and Sharp, O. J.: Compiler Transformations for High-Performance Computing, *ACM Computing Survey*, Vol. 26, No. 4, pp. 345–420 (1994).
3) Banerjee, U.: *Dependence Analysis for Supercomputing*, Kluwer Academic Publishers (1988).
4) Banerjee, U.: *Loop Transformations for Restructuring Compilers: the foundations*, Kluwer Academic Publishers (1993).
5) Hall, M., Harvey, T. J., Kennedy, K., McIntosh, N., McKinley, K., Oldham, J. D., Paleczny, M. H. and Roth, G.: Experiences Using the ParaScope Editor: an Interactive Parallel Programming Tool, *SIGPLAN Notice*, Vol. 28, No. 7, pp. 33–43 (1993).
6) 岩沢京子, 黒澤隆, 菊池純男: 並列化支援システムによる Fortran DO ループの並列化方法, 情報処理学会論文誌, Vol. 36, No. 8, pp. 1995–2006 (1995).
7) Polychronopoulos, C.: *Parallel Programming*

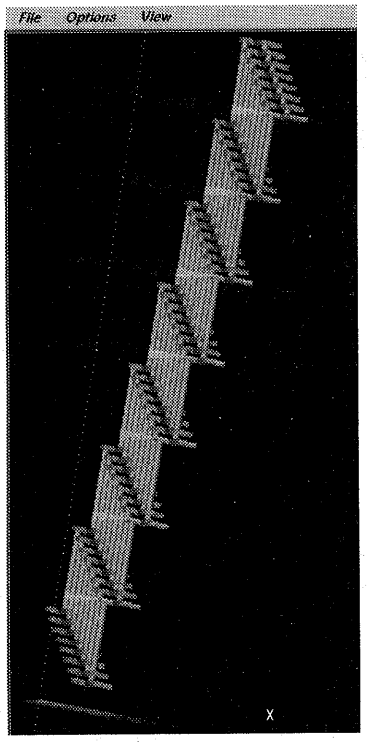


図6 図3のデータ依存関係（W-Rポール）
Fig. 6 Visualizaton of data dependence of Figure3(with W-R pole)

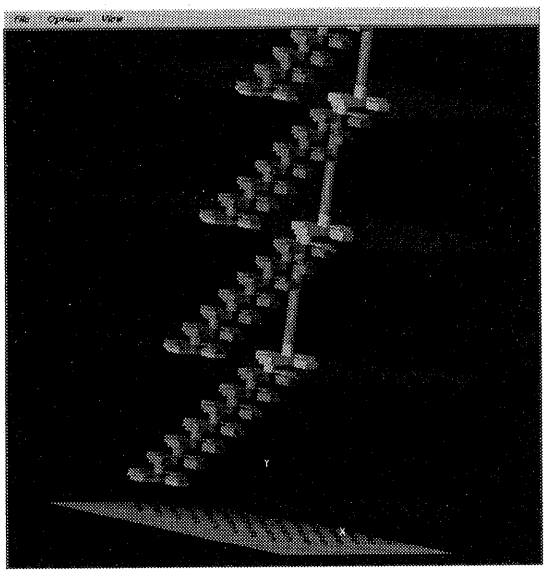


図7 図3のデータ依存関係（最も短いW-Rポール）
Fig. 7 Visualizaton of data dependence of Figure3(with the shortest W-R pole)

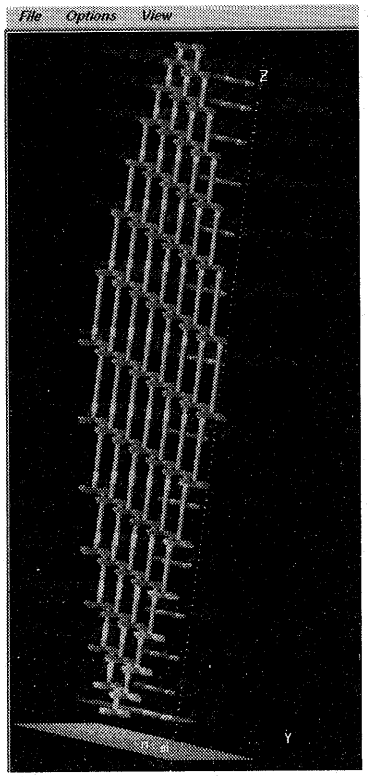


図8 図3にループスキューイングを施して得られたデータ依存関係（W-Rポール）
Fig. 8 Visualizaton of data dependence of skewed Figure3(with W-R pole)

and Compilers, Kluewer Academic Press (1988).

8) Sasakura, M., Joe, K. and Araki, K.: NaraView: An Interactive 3D Visualization System for Parallelization of Programs, *Int'l Symp. on High Performance Computing '97, Lecture Notes in Computer Science 1336* (Polychronopoulos, C., Joe, K., Araki, K. and Amamiya, M.(eds.)), Springer, pp. 231–242 (1997).

9) 下村隆夫: プログラムスライシング技術と応用, 共立出版株式会社 (1995).

10) Zima, H. and Chapman, B.: *Supercompilers for Parallel and Vector Computers*, Addison-Wesley (1991). (村岡洋一訳: スーパーコンパイラオーム社 (1995)).

11) 笹倉万里子, 木和田智子, 城和貴, 荒木啓二郎: 並列化手法選択支援のためのループ内データ依存 関係の3次元視覚化法, 情報処理学会並列処理シンポジウム JSPP '97, pp. 297–304 (1997).

(平成10年7月2日受付)
(平成10年8月13日再受付)
(平成10年9月4日採録)

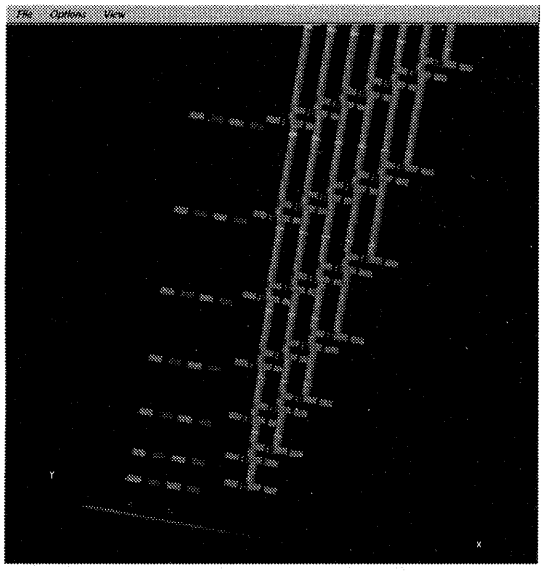


図9 図3にループスキューイングを施して得られたデータ依存関係（最も短いW-Rポール）
Fig. 9 Visualizaton of data dependence of skewed Figure3(with the shortest W-R pole)

笹倉万里子（正会員）

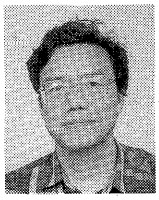
京都大学工学部情報工学科卒，（財）京都産業情報センター，（財）京都高度技術研究所勤務を経て，1995年 奈良先端科学技術大学院大学情報科学研究科博士前期課程修了．1996年 同大学院大学情報科学研究科博士後期課程中退．同年岡山大学工学部情報工学科助手．現在に至る．修士（工学）．並列化支援のための視覚化システムに関する研究に従事．情報処理学会，日本ソフトウェア科学会，人工知能学会，IEEE-CS 各会員．

城　和貴（正会員）

阪大・理・数学卒．日本DEC，ATR視聴覚研究所（日本DECより出向），（株）クボタ・コンピュータ事業推進室で勤務．1993年奈良先端科学技術大学院大学情報科学研究科博士前期課程入学，1996年同研究科後期課程修了，同年同研究科助手．1997年和歌山大学システム工学部情報通信システム学科講師，1998年同学科助教授．工博．画像処理，文字認識，ニューラルネットワーク，並列計算機アーキテクチャ，自動並列化コンパイラ，並列計算機の解析モデル，視覚化などの研究に従事．IEEE会員．

國枝　義敏（正会員）

1980年 京大・工・情報工学科 卒業，1982年 同大大学院・工・情報工学専攻・修士課程了，同年 同大・工・情報工学科 助手，1991年 同 助教授，1996年 和歌山大学・システム工学部・情報通信システム学科 教授，1998年 同大・システム情報学センター 教授，同センター 副センター長，現在に至る．京大工学博士．並列処理，分散処理，基本ソフトウェア，プログラミング言語，ハイパフォーマンスコンピューティングの研究に従事．情報処理学会，電子情報通信学会，ACM, IEEE-CS 各会員．1992年 情報処理学会論文賞授賞．



荒木啓二郎（正会員）

1954年 福岡市生まれ．1976年 九州大学工学部卒業．1978年 同大学院修士課程修了．九州大学工学部助手，九州大学工学部助教授，奈良先端科学技術大学院大学情報科学研究科教授を経て，現在，九州大学大学院システム情報科学研究科教授．（財）九州システム情報技術研究所研究室長兼務．工学博士．プログラミング言語，形式的仕様記述，並列／分散処理，インターネット，マルチメディア通信などの研究に従事．ソフトウェア技術者協会幹事，博多祇園山笠西流赤手拭など．