ハイパフォーマンス
コンピューティング 93－20
（２００３．３．１２）

並列化チューニングツールAiviにおける手続き間データ依存位置検出機能の開発

佐藤 真琴[1),2)] 和田 清美[1),2)]
1) 日立製作所システム開発研究所 2) アドバンスト並列化コンパイラ研究体

要旨

マルチプロセッサ上で最高性能を得るにはユーザの知識を利用した並列化チューニングが必須である．従来，ループ並列化を阻害するデータ依存関係を持つ文の対を検出するツールがあったが，これは一方の文が手続き呼び出しの場合，呼び出された手続き内のどの文とデータ依存関係にあるかまでは検出できなかった．我々はこのような文をオンデマンドに検出するツールを開発した．本ツールは，また，解析対象ループからデータ依存関係にある文に至る手続き呼び出し列やデータ依存情報の表示，データ依存関係にある文のソースプログラム上での表示が可能である．以上により，手続き呼出しを含むループに対する並列化チューニング作業の効率化が期待できる．

Development of Interprocedural Data-dependence Locator in Parallel Tuning Tool Aivi

Makoto Satoh[1), 2)] Kiyomi Wada[1), 2)]
1) Systems Development Laboratory, Hitachi, Ltd. 2) Advanced Parallelizing Compiler Project

Abstract

To obtain highest performance on multiprocessors parallelization tuning using user's knowledge is indispensable. There are some tools that show pairs of statements that have data-dependence relationship prohibiting parallelization of a loop. These tools, however, can not show any statement having data-dependence relationship in a procedure called within the loop. We have developed a tool that finds all those statements. Our tool can show a call sequence starting from the loop in question to the statement having data-dependence relationship, the data-dependence information, and two source programs, each of which includes either of the pair of statements having data-dependence relationship. Using this tool we can expect the parallelization tuning to be done efficiently for loops including procedure calls.

1. はじめに

本研究の目的は，手続き呼び出しを含むループに対して，呼び出される手続き内にある，データ依存関係にある文を自動的に検出するツールを開発することである．

プログラミングの容易さと広範囲のプログラムで実行性能向上が期待できる点で利用範囲が拡大している共有メモリ型マルチプロセッサ（Shared-Memory Multiprocessor: SMP）上で，Fortran等の既存の言語で書かれたプログラムを実行するため，我々は，プログラムを自動的に並列化する手続き間自動並列化コンパイラ WPP（Whole Program Parallelizer）の研究開発を進めている[1,2]．ところが，プログラムの並列化判定には動的情報が必要な場合があるため，静的情報のみを利用する自動並列化コンパイラでは限界がある．また，プログラム実行中に並列化可否を判定し，可能であれば並列実行するようなコード生成方法もあるが，実行時オーバヘッドが発生する．そこで，ユーザの知識を利用して並列化を行なう並列化チューニングが重要になる．

並列化チューニングでは自動並列化コンパイラが並列化不可と判定したループを対象とするため，その判定理由，特に並列化を阻害する可能性があるデータ依存関係を持つ文の対（データ依存元とデータ依存先），をユーザに表示することが出発点となる．ユーザは，このような文の対の間に，実際はデータ依存関係がないことがわかるなら，並列化可能と判定できる．

このような文の対の表示を行なうツールとしては，従来，ループ内でデータ依存関係の可能性のある２つの文を表示するツールがあった[4-6]．しかし，一方の文が手続き呼び出し文の場合，これらのツールは，呼び出された手続き内のどの文とデータ依存関係にあるかを表示しないため，ユーザはその手続きを自ら解析して依存の有無を調べる必要があり，チューニング作業は非常に困難であった．

そこで，我々は，並列化解析情報ビジュアライザAivi（Analysis Information Visualizer）を開発し，手続き

呼び出し文やループレベルで配列参照範囲を表示している．ユーザは，データ依存関係にある２つの文を含む手続き呼び出しまたはループに対して，同じ配列の参照範囲をユーザ自身が比較することにより，手続き境界をまたがるデータ依存の有無が手続き呼び出し文やループレベルで確認できる[3]．

本研究はこのAiviの機能を強化し，手続き境界をまたがるデータ依存の位置を文レベルで自動的に検出するツールを開発した．本ツールは解析対象ループと同じ手続き内にある，そのループに対してデータ依存関係にある文の対を求めるループ内全データ依存位置検出機能と，それらの文の対のうち，少なくとも一方が手続き呼び出し文の時，そのデータ依存先（元）が，呼び出された，その手続き内の，どの文であるかを求める手続き間段階的データ依存位置検出機能から成る．

本論文の構成は以下となる．まず，２章で機能概要と構成を述べ，３章でデータ依存解析の実現方法を説明する．４章ではループ内全データ依存位置検出機能を，５章では手続き間段階的データ依存位置検出機能を説明し，６章ではツールのユーザインターフェースについて述べる．７章で関連研究を述べ，８章で結論を述べる．

2. 機能概要と構成

2. 1 機能概要

図１は手続き間データ依存位置検出機能を説明した図である．

図１(a)は従来のツールによるデータ依存位置検出機能が表示可能なデータ依存位置である[4-6]．解析対象ループLと同じ手続き（ベース手続きと呼ぶ）内にある，ループLに関して運搬依存のある文が表示される．

図１(b)は手続き間データ依存位置検出機能が表示可能なデータ依存位置である．ベース手続き内の文S1とベース手続き内から呼び出される手続き内にある，ループLに関して文S1とループ運搬依存のある文Qが表示される．

2. 2 Aiviの特徴

手続き間データ依存位置検出機能が実装されたツールAiviは以下の３つの特徴を持つ．

（1）データ依存元と依存先の段階的な表示
（2）データ依存元と依存先の別画面上での表示
（3）データ依存元と依存先の関連付け情報の表示

以下，これらを説明する．

（1）データ依存元と依存先の段階的な表示

これは解析対象ループ内の全てのデータ依存位置（ループから呼び出しされる手続き内にあるものも含む）を一度に表示するのではなく，ユーザ指定により段階的に詳細な情報を表示することを意味する．即ち，まず，図１(a)のようなベース手続き内の全てのデータ依存位置を表示し，次に，文S2をユーザが選択した時にのみ，図１(b)のように呼び出し先手続き内でのデータ依存位置を表示する．

これにより，一度に過剰な情報を表示せず，ユーザが注目するデータ依存情報のみ表示するので，ユーザは快適に作業を続けることが期待できる．

（2）データ依存元と依存先の別画面上での表示

これは以下の２つを意味する：

(a) データ依存元と依存先に対するデータ依存情報を別画面で表示．
(b) データ依存元と依存先をソースプログラム上，別画面で表示

(a)により，データ依存元と依存先の両者が手続き呼び出し文であっても，ユーザは一方だけを選択して段階的表示が可能となり，過剰な情報の表示が避けられる．

また，(b)により，データ依存元と依存先が異なる手続きにあるなど，両者がプログラム上かなり離れた位置にある場合でも両者の同時表示が可能なのでユーザによる作業の効率化が期待できる．

（3）データ依存元と依存先の関連付け情報の表示

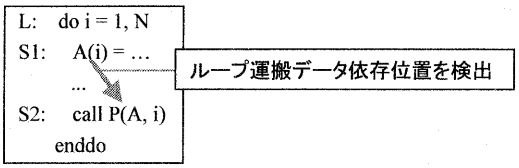
これは解析対象ループからデータ依存元と依存先に至る手続き呼び出し文を，各々，表示することにより，両者の関連付けを明確化する機能である．

これにより，データ依存元と依存先が共にベース手続きとは異なる手続き内にある場合に，両者間にデータ依存が実際にあるか否かを確認する時，調査すべき手続きや手続き内の文の範囲が明確となるので，ユーザによる作業の効率化が期待できる．

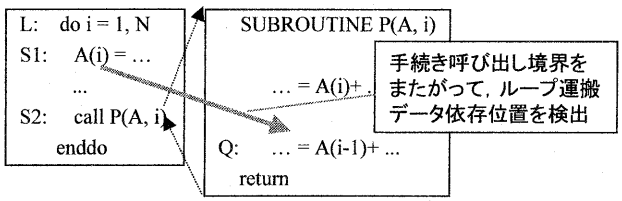
2. 3 構成

図２は，Aiviの構成である．以下，ユーザによる本ツールの操作とツール内部の処理の流れを説明する．

（1）WPPコンパイラはソースプログラムを入力して手続き間解析及び手続き間並列化を適用し，Aiviは得られたループ解析情報をユーザに表示する．
（2）ユーザはWPPが並列化しなかったループのうちの１つを解析対象ループLとして指定する．
（3）WPPはベース手続き内の全てのデータ依存位置を解析してその結果を選択履歴・データ依存情報ファイル（履歴ファイルと略す）に出力し，Aiviはこれを入力してデータ依存位置リスト表示を行なう．
（4）データ依存位置の一方となる，ループL中の手続きPに対する呼出しをユーザが指定すると，Aiviはその情報を履歴ファイルに出力し，WPPを起動する．
（5）WPPは履歴ファイルを入力して手続きP内におけるデータ依存位置情報を解析して，その結果を履歴ファイルに出力し，Aiviはこれを入力してデータ依存位置リスト表示を行なうと共に，選択履歴表示を行なう．
（6）ユーザがデータ依存位置リスト表示中の，１組のデータ依存元と依存先を選択してソース表示機能を指定すると，Aiviは両者が含まれるソースプログラムを各々，別画面上に表示する．



(a) 従来のデータ依存位置検出機能



(b) 手続き間データ依存位置検出機能

図１： 手続き間データ依存位置検出機能

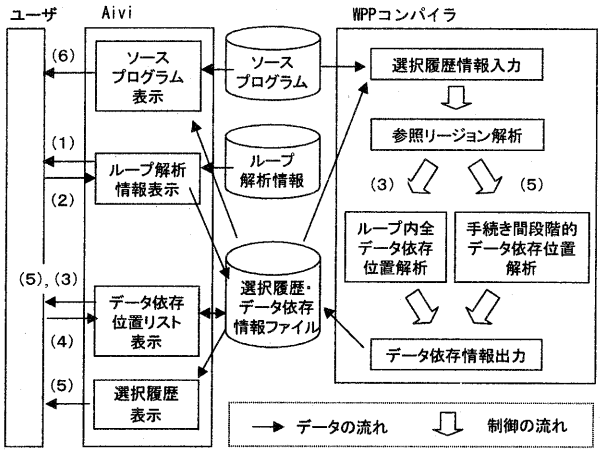


図２：手続き間依存位置表示ツール Aivi の構成

表１： 段階的表示機能の実現方式の比較

	解析データ量	解析時間		
		初回	ユーザ選択時	再解析時
一括解析	×	×	○	×
段階的解析	△	△	△	△

（1）対象ループ内のプログラムに対する階層的制御フローグラフ（CFSと呼ぶ）を作成する.
（2）CFSの全ノードに対して配列リージョンを解析して保持する.
（3）最上位階層のノード間のデータ依存を解析し，その結果からデータ依存ノードリストを作成する.
（4）データ依存ノードリストをたどって，第2階層以下のノード間のデータ依存をリカーシブに解析し，その結果からデータ依存ノードリストを更新する.
（5）最下層のノードからなるデータ依存ノードリストをたどり，そのノード対に含まれる各文の配列リージョンを解析して比較し，文レベルでのデータ依存位置を決定する.

図3：ループ内全データ依存位置検出アルゴリズム

3. データ依存の解析方法

データ依存位置の検出は，対象とする各文に対する配列リージョンを比較して行なう．ある配列に対する配列リージョンとは，あるプログラム単位内で，その配列がある種の参照を行なう時の配列添字の集合である．我々は配列Aに対して以下の4種類を解析している.

MOD_A：	定義される可能性の有る添字範囲
USE_A：	使用される可能性の有る添字範囲
$KILL_A$：	必ず定義される添字範囲
$EUSE_A$：	定義前に使用される可能性の有る添字範囲

これらを用いて，例えば，ループ運搬フロー依存があることを示すには以下を示せば良い：

$$MOD_A[1:i-1] \cap EUSE_A[i:i] \neq \phi. \qquad (1)$$

ここで，iは一般の値を取る変数を表わし，$MOD_A[1:i-1]$はループ繰り返し1回目から(i-1)回目の間に定義される可能性の有る添字範囲を表わす．$EUSE_A$についても同様である．式(1)がループ運搬フロー依存を示すのは以下のように説明できる．まず，次の式(2)は依存距離が1のループ運搬フロー依存を示す.

$$MOD_A[i-1:i-1] \cap EUSE_A[i:i] \neq \phi. \qquad (2)$$

したがって，式(1)は依存距離が1からi-1までの範囲のいずれかのループ運搬フロー依存があることを示す．iは一般の値を取る変数なので，式(1)はいずれかの依存距離を持つループ運搬フロー依存を示す.

4. ループ内全データ依存位置検出機能

本機能の実現方式として，以下の2つを検討した：

(a)全文対解析方式
(b)階層的解析方式

(a)はループ内の全ての文と文の対の配列リージョンを比較して，データ依存の有る文の対を検出する方式である.

(b)は階層的制御フローグラフを作成し，データ依存のあるノードの対を，上位階層から下位階層に順番に検出していく方式である.

（1）履歴ファイルを入力し，ユーザが指定した手続き，及び，その手続きの呼び出し文とデータ依存関係に有る文に対してフラグを立てる.
（2）フラグの立った手続きに対しては，それに含まれる全ての文に対する配列リージョンを解析する．また，フラグの立った文に対しては，その文に対する配列リージョンを解析する.
（3）（2）で得られた配列リージョンに対して，解析対象手続きに到達するまで，以下を適用する.
・入口点伝播（配列リージョンを手続き入口点における変数の値を用いて表わす処理）
・ループ入口点伝播（ループが全繰り返し回だけ実行した時の配列リージョンに変換し，それをループ入口点における変数の値を用いて表わす処理）
・呼び出し元伝播（呼出し先手続きの変数名を，呼出し元手続きの変数名に変換する処理）を適用する.
（4）得られた配列リージョン同士を比較し，データ依存を持つ文の組を特定する.

図4：手続き間段階的データ依存位置検出アルゴリズム

我々は(b)の方式を採用した．この方式の方が配列リージョンの比較回数が少なくなると期待できるからである．図3はこの方式のアルゴリズムを示す.

5. 手続き間段階的データ依存位置検出機能

5.1 実現方式の検討

本機能の実現方式として，以下の2つを検討した：

(a)一括解析方式
(b)段階的解析方式

(a)は解析対象ループ中の全てのデータ依存位置を，全手続き呼び出しに対して予め一括して解析しておき，ユーザにはその解析結果をフィルタリングして，ユーザが選択した手続きの情報のみ表示する方式である.

(b)は解析対象ループ中の，ユーザが選択した手続きに対してのみデータ依存位置をその都度，解析し，その結果をユーザに表示する方式である.

表1は両実現方式の解析データ量と解析時間を比較したものである．表中にある再解析とは，ユーザが，変数の値の範囲や変数間の関係式等を表わす指示文をソースプログラムに挿入した場合等に発生する解析を指す．一括解析は，初回や再解析時には，解析に多くの時間がかかるが，ユーザ選択時には，ユーザ指示に従って解析結果を表示するだけなので，レスポンスは良い．一方，段階的解析は，初回や再解析時には，ループ内の，ループと同じ手続き内にあるデータ依存のみ解析するため，解析時間はあまりかからないが，ユーザ選択時にも解析を行なう必要がある.

我々は(b)の方式を採用した．その理由は以下である.
・段階的解析はデータ量が少ない.
・一括解析ではユーザが選択してない手続きに対する解析

結果は無駄になる.

・段階的解析ではユーザ選択時のレスポンスを改良する対策があり得る．例えば，コンパイラを終了させず解析部直前で実行待ちをし，メモリ上に展開された中間語を再利用する，などが考えられる.

図４はこの方式のアルゴリズムを示す.

6. ユーザインターフェース

以下では主要なユーザインターフェースの機能を述べる．図５の例題プログラムに対するウィンドウを図６と図７を使って説明する.

6.1 データ依存位置リスト表示ウィンドウ

図６は本ウィンドウの例である．本ウィンドウは，主に次の３種類のサブウィンドウから成る.

（1）ユーザ選択履歴表示用サブウィンドウ

ユーザが手続き呼び出しをたどってデータ依存元やデータ依存先を表示した履歴を，コールグラフ形式でグラフィカルに表示する．これにより，データ依存元と依存先との間にデータ依存関係があるか否かを確認するために調査すべきプログラム範囲がわかりやすく表現されるため，作業効率の向上が期待できる．図６では，メインプログラムから呼び出される DEFA と USEA 内のデータ依存をユーザが表示したことを示し，現在，ユーザが選択しているのは DEFA 内の文であることを強調表示している.

（2）データ依存情報表示用サブウィンドウ

データ依存の種別やデータ依存の確度を表示する．図６の第２行目は，ループ運搬依存(LC)，フロー依存(FLOW)，依存は確定的でない（INDEFINITE）ことを示す.

（3）データ依存元（依存先）表示用サブウィンドウ

データ依存元または依存先となる文に対して，それが含まれる手続き名，行番号，配列リージョンを表示する．図６の強調表示されている行はDEFA 中の１７行目の文の配列 X のリージョンは X(i-M) であることを示す.

6.2 ソースプログラム表示ウィンドウ

図７は本ウィンドウの例である．図６の中の１組のデータ依存位置を選択することにより，その文を含むプログラムが表示され，データ依存元及び依存先が強調表示される.

6.3 ウィンドウ上での操作機能

以下の隠蔽・表示機能と表示並べ替え機能がある.

（1）隠蔽・表示機能

データ依存位置リスト表示において，以下の２つの単位での行の隠蔽・表示ができる.

- 行単位の隠蔽・表示
- 手続き単位の隠蔽・表示

これらにより，注目している依存情報を連続的に並べたり，すでにチェック済みの依存情報の隠蔽が可能となる.

（2）表示並べ替え機能

データ依存位置リスト表示において，以下の２つの並べ替えができる.

- 依存元順並べ替え
- 依存先順並べ替え

ここで，依存元（先）順並べ替えは，同じデータ依存元（先）を持つデータ依存先（元）が連続するように並べ替える．これによって，ユーザは１つの依存元または依存先を固定して，それとデータ依存関係にある文をチェックできるので，作業効率の向上が期待できる.

7. 関連研究

Rice 大の ParaScope Editor[4]はデータ依存元と依存先をリスト表示し，ソースプログラム上でそのデータ依存を矢印で表示する．また，両方のデータ依存位置が離れている場合にはソースプログラムの一部を隠蔽して，それらを１画面に表示することが可能である.

Stanford 大の SUIF/Explorer[6]はある文で参照される変数に対して，その変数値に影響を与える可能性のある文から成るプログラムスライスを表示する．また，データ依存元と依存先をソースプログラム上で強調表示する.

CAPTools[5] は，データ依存元と依存先をノードとする依存グラフを表示する.

以上のいずれにおいても手続きをまたがるデータ依存位置の検出に関する記述はない.

8. おわりに

ループ運搬依存を持つ２つの文の位置を，それらの一方または双方が，ループ内から呼び出される手続き内にある場合でも，ユーザ指示により段階的に表示する手続き間データ依存位置検出機能を開発した．今後はベンチマークプログラムへの適用を行なう予定である．尚，本研究の一部は，経済産業省/NEDO ミレニアムプロジェクト「アドバンスト並列化コンパイラ技術」により行なわれた.

参考文献

[1] 青木等：手続き間自動並列化コンパイラ WPP の試作 －実機性能評価－，情処研究報告，98-ARC-130，pp. 43-48 (1998).

[2] 佐藤等：手続き間自動並列化コンパイラ WPP の評価，第 130 回計算機アーキテクチャ研究会 (SHINING 2001 (2001)).

[3] 佐藤等：コンパイラ解析情報ビジュアライザ Aivi，情処第 62 回全国大会，4R-05，Mar.，(2001).

[4] M. Hall et al. "Experiences using the ParaScope Editor: an Interactive Parallel Programming Tool", PPoPP '93,

[5] S. P. Johnson et al. "Computer Aided Parallelization Tools (CAPTools) User Manual, CAPTools Version 2.0Beta", http://captools.gre.ac.uk, Oct. 1998.

[6] S. -W. Liao et al. "SUIF Explorer: An Interactive and Interprocedural Parallelizer", PPoPP'99, pp. 37-48, 1999.

```
 1   program apc                          14   subroutine defa (X, N, i, M)
 2   parameter (N=1000)                   15   integer X(N)
 3   integer A(N), B(N)                   16   integer N, M, i
 4   integer M                            17   X(i- M) = i - M
 5   read (5, *) M                        18   return
 6   do i = 2, N-1                        19   end
 7     A(i) = i
 8     call defa (A, N, i, M)             20   subroutine usea (X, Y, N, i)
 9     call usea (A, B, N, i)             21   integer X(N), Y(N)
10   enddo                                22   integer N, i
11   write (6,*) B (N-1)                  23   Y(i) = X(i) + 1.0
12   stop                                 24   return
13   end                                  25   end
```

図５：　例題プログラム

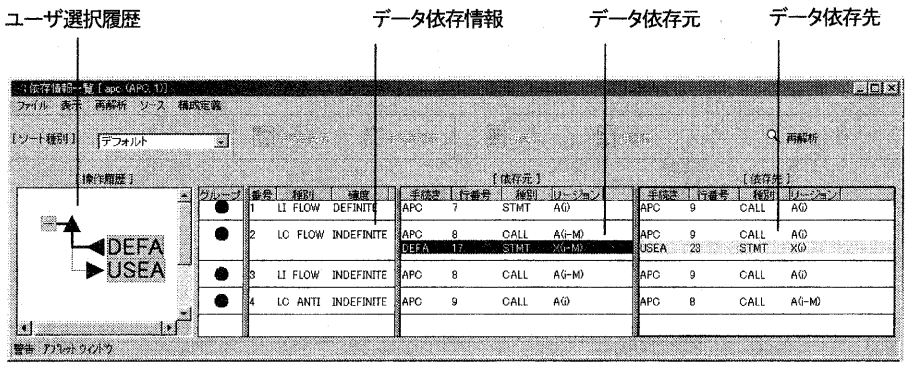


図６：　データ依存位置リスト表示ウィンドウ

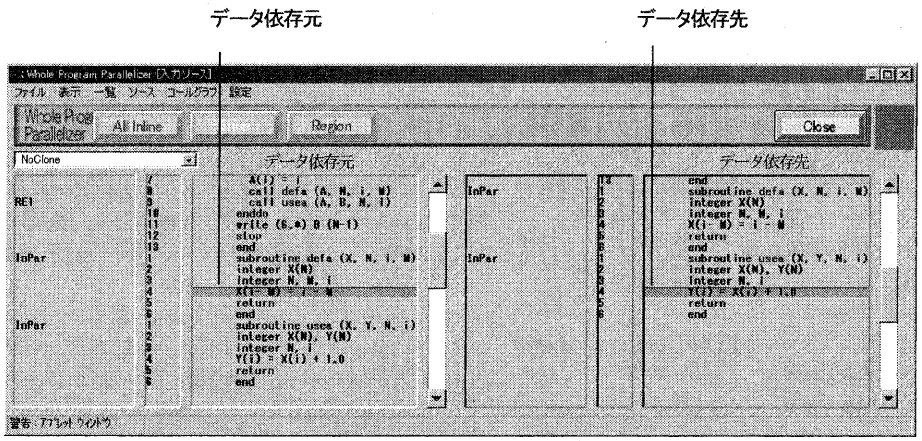


図７：　ソースプログラム表示ウィンドウ