シンプレックス法に基づく実用的な配列データ依存解析

峰尾昌明[†1] 上原哲太郎[†2]
齋藤彰一[†3] 國枝義敏[†4]

自動並列化コンパイラにとって，並列実行可能性を判別するためにデータ依存解析モジュールは必須である．配列要素間のデータ依存解析手法は種々提案されており，各手法には解析の速度と厳密性との間にトレードオフがある．厳密性を重視した依存解析手法として Omega テストが有名である．しかし，Omega テストは，解析にかかる時間が長く，また実装が困難である．本論文では，実装が容易かつ，Omega テストとほぼ同程度の厳密性を持ち，多くの場合 Omega テストより，高速に解析を行う新たな手法を提案する．本手法は，線形計画法と全探索を組み合わせ，さらに，GCD テスト，Banerjee テスト，分離テストの機能をも取り込んだ新しい独自の統合的アルゴリズムとなっている．本論文では，この統合的アルゴリズムの詳細，実装および，性能評価について報告する．

A Practical Test for Array Data Dependence Analysis Based on Simplex Method

Masaaki Mineo,[†1] Tetsutaro Uehara,[†2] Shoichi Saito[†3]
and Yoshitoshi Kunieda[†4]

Data dependence analysis is essential for automatic parallelizing compilers. Compilers determine the possibility of parallelization on given source programs by using the result from data dependence analysis. Several dependence analysis tests on array data have already been proposed. Each test cannot avoid the trade-off between its analysis speed and exactness of analysis. Among conventional tests, Omega test is well known as an exact test for the broader class of index expressions ever. However, the algorithm of Omega test is so complicated that its analysis is very time consuming and it is difficult to implement Omega test. Therefore, in this paper a new original analysis method is proposed, whose algorithm is based and combined both Simplex method for linear programming and exhaustive solution search method. This algorithm also includes the features of GCD test, Banerjee test, and Separability test. The algorithm, implementation details of prototype and its evaluation applying to the concrete test programs are also described.

1. はじめに

プログラムの並列化を行う際に主な対象となるものはループである．その理由は，通常，ループには高い並列性と適度な並列粒度が期待できること，プログラムの実行時間の中でループ部分が占める割合が大きいこと，ループの並列実行可能性解析が比較的行いやすいことなどがあげられる．しかし，ループ内の変数に依存関係がある場合，ループボディごとに並列処理させるとループ繰返しの順序が変化することにより，実行結果が逐次実行の結果と異なることがある（厳密には 2 章を参照）．よって，並列化を行うときはループ内の変数の依存関係を調べるために，依存解析を行う必要がある．

配列データの依存解析手法はすでに種々提案されており，解析の速度と厳密性との間にトレードオフがある．速度を重視する手法として GCD テスト[1] と Banerjee テスト[2] がよく利用される．他方，厳密な手法として Omega テスト[3] がよく知られている．Omega テストは，解析にかかる時間が膨大[4),5)] である．しかし，一般的なプログラムに対しては Banerjee テストだけでほぼ解析が可能であること，Omega テストの厳密な解析が必要となるのは，複雑な数値計算プログラムに

†1 和歌山大学大学院システム工学研究科
Graduate School of Systems Engineering, Wakayama University
†2 京都大学工学研究科附属情報センター
Center for Information Technology, Faculty of Engineering, Kyoto University
†3 和歌山大学システム工学部
Faculty of Systems Engineering, Wakayama University
†4 立命館大学情報理工学部情報システム学科
Department of Computer Science, College of Information Science and Engineering, Ritsumeikan University

```
do i = 1,n
   A(i+1) = B(i) + C(i)    :S1
   C(i)   = A(i) + B(i)    :S2
end do
```

図 1 プログラム例 1
Fig. 1 Program example 1.

```
do i = 1,n
      A(2*i+2) = B(i) + C(i)    :S1
      C(i)     = A(3*i) + B(i)  :S2
end do
```

図 2 プログラム例 2
Fig. 2 Program example 2.

対してのみといえることが知られている[4),5)]．

本論文では Omega テストと比較して，(1) ほぼ同程度の厳密な解析能力を有し，(2) 解析時間が平均して短く，(3) 実装が容易である，という特徴を有する新しい解析手法を提案する．本手法のアルゴリズムは，線形計画法の解法の 1 つであるシンプレックス法[6)] と整数解の全探索に基づき，さらに GCD テスト，Banerjee テスト，分離テストの機能をも取り込んでいる．このことから，本手法を **Laputa**（**L**ine**A**r **P**rogramming and exha**U**s**T**ive se**A**rch）テストと名付けた．

以下，2 章でデータ依存関係解析について述べ，3 章で関連研究をまとめ，4 章で Laputa テストのアルゴリズムを紹介する．5 章で Laputa テストと Omega テストの解析速度と精度を測定する．6 章では 5 章の Laputa テストと Omega テストの測定結果に関して考察する．最後に 7 章でまとめを述べる．

2. データ依存関係解析

本章では準備として用語の定義について述べた後，問題を定式化する．本章の内容は，文献 1)，2) を一部要約した．

2.1 データ依存

プログラム中の同一の変数，あるいは配列要素に対する二参照の実行順序を変更することにより，実行結果が異なる可能性がある関係のことを**データ依存**と呼ぶ．さらにループについては，繰返しを考慮する必要がある．例として，**図 1** のプログラム例 1 の代入文 S1 の配列参照 $\mathbf{A(i+1)}$ と代入文 S2 の配列参照 $\mathbf{A(i)}$ に注目すると，$\mathbf{x}$ 回目（$1 \leq x \leq n-1$）の繰返しの S1 で配列 $\mathbf{A(x+1)}$ に書き込まれたデータを $\mathbf{x+1}$ 回目の S2 で引用することから，ループの繰返しによる依存があることが分かる．異なった繰返し間でのデータ依存を**ループ繰越し依存**と呼ぶ．これに対し，S1 の配列 $\mathbf{C(i)}$ と S2 の配列 $\mathbf{C(i)}$ のように同一の繰返しにおいて生じるデータ依存を**ループ独立依存**と呼ぶ．

以後，依存関係が存在することを “依存あり”，存在しないことを “依存なし”，不確かな場合を “依存の可能性あり” と書く．

2.2 依存の方向と距離

一般にループの繰返しの $\mathbf{i_1}$ 回目と $\mathbf{i_2}$ 回目との間にループ繰越し依存が生じる場合，$\mathbf{i_1}$ と $\mathbf{i_2}$ の大小関係により，式 (1) の 3 種類の**依存方向**に分類でき，それぞれ “>”，“=”，“<” で書き表す．さらに，$|\mathbf{i_1} - \mathbf{i_2}|$ を**依存距離**と定義する．

$$\begin{cases} \mathbf{i_1} - \mathbf{i_2} > \mathbf{0}：\text{正} \\ \mathbf{i_1} - \mathbf{i_2} = \mathbf{0}：\text{零} \\ \mathbf{i_1} - \mathbf{i_2} < \mathbf{0}：\text{負} \end{cases} \tag{1}$$

依存方向，依存距離の概念は，多重ループ内のデータ依存についてもそのまま拡張できる．依存方向と依存距離の情報を用いて，ループ再構築技法[1)] を適用することにより，並列化が可能となる場合があるため，これらの情報は重要である．

2.3 配列データ依存解析問題の定式化

図 2 のプログラム例 2 の配列 A に対する二参照について考える．2.1 節で述べたループ繰越し依存およびループ独立依存の両方を解析するためには，代入文 S1 の配列 $\mathbf{A}$ の添え字式と代入文 S2 の配列 $\mathbf{A}$ の各添え字式について，同一の制御変数 $\mathbf{i}$ が独立に可変であるとき（$\mathbf{i_1}$，$\mathbf{i_2}$ とする）に，二参照の添え字式が等しくなるか否かを調べればよい．

以上より，次式 (2) の方程式を満たす整数解 $\mathbf{i_1}$，$\mathbf{i_2}$ が，ループの上下限から求まる制御変数としての定義域内（式 (3)，以下，単に「上下限式」と記す）に存在するかという問題に定式化できる．すなわち，方程式 (2) と不等式 (3) からなる「系」の全条件を満たす整数解を求解する．

$$\mathbf{2i_1 + 2 = 3i_2} \tag{2}$$

$$\mathbf{1 \leq i_1, i_2 \leq n} \tag{3}$$

式 (2) が整数解を持つ問題は，一般に線形ディオファンタス方程式と呼ばれるが，以降では混乱しない限り，単に（ディオファンタス）方程式と呼ぶ．また，一般に k 次元配列の場合は各次元ごとにディオファンタス方程式を 1 つ作成し，まとめて k 次連立方程式として解析すればよい．ループ構造を一般化して考えると，多重ループに属する場合には，添え字式はループ制御変数の線形式であることを前提とする．さらに各ループ上下限式は，より外側のループの制御変数を含む線形式であるものとする．最外側ループの上下限式は整定数とする．

3. 関連研究

文献 7) では，高速な Extended GCD Test[8] をはじめ種々の方式を組み合わせ，解析精度と解析速度の両面を追求した依存解析法を新たに提案している．しかし，厳密性を確保するために，核としているのは，Omega テストが基礎としている同じ Fourier-Motzkin アルゴリズム[9] を採用している点で，我々の提案手法とは異なる．実際に Perfect Club Benchmarks（以下，PB）[10),11)] を使い依存解析手法の性能評価をしている．

文献 12) では，依存解析手法のうちで，理論的に最も厳密な手法の代表としてシンプレックス法をベースとした整数計画法および Omega テストの 2 種，ならびに，これらほど厳密ではないが，解析速度の速い，ある種の近似を用いる Constant Test[1]，GCD test，Banerjee test の 3 種，計 5 種を取り上げ，PB，LAPACK[13]，EISPACK[14] を用いて，実際に詳細に比較検討している．さらに，特徴的な評価法として，実行時に収集される様々な情報も使って，各手法の厳密性を比較している．

この文献で用いられているシンプレックス法ベースの手法では，同法 1 回の適用で整数解の存在判定ができなかった場合には，いわゆる Branch and Bound により，小さな部分問題に分割して，繰り返し同法を適用する，繰返し型（収束するまで）のアルゴリズムを採用している．これは，別の見方をすると，解空間を分割しつつ整数解が存在することが判明するか，整数格子点を含まないほど小さくなるまで，分割を繰り返していることにほかならない．通常シンプレックス法，整数計画法および線形計画法（以下シンプレックス法類と呼ぶ）を利用する場合，一般には 1 回だけでは厳密に求解できないため，反復解法とならざるをえないという認識であると思われる（後述の他論文でも同様の記述が見られる）．この認識の下では，シンプレックス法類に基づく厳密な解析は非効率的であると予想されるのは当然である．これに対し，次章で詳述する我々の提案手法は反復解法ではなく，全探索を行い，非常に単純に総当たりしようとするもので，その結果，多くの場合においては反復解法よりも高速に解析が可能であることを本論文で示そうと考えている．

以下，シンプレックス法類を適用する類似研究について，比較検討する．

文献 15) では，シンプレックス法に基づく一般に適用可能な依存解析手法数種を提案している．この文献の 2 章の冒頭でまず，「整数計画法（および線形計画法）の十分性問題に関する計算の複雑さは，一般に言われているほど，“so difficult” ではない．特に，この依存解析に適用する場合には，通常とは各段に変数の数が少ないからである」とコメントされている．ここで提案されている厳密な解法は，前処理としてまず，Gauss 消去，GCD テスト，不等式群に選択的 Fourier-Motzkin を必要なら反復的に適用したうえで，最終的に整数計画法の解法には，Rudimentary Primal All-Integer Algorithm（RPAI），Constraint-Matrix test[16]，“FAS3T”，Simple Dual All-Integers test，Surrogate Dual All-Integers test の 5 種のいずれか 1 つを使う．これら 5 種はいずれも，基本的には最初の RPAI を改変したもので，変数消去（Gauss）のピボット選択時，1 以外の係数の場合に “cut” と呼ばれる新たな不等式が追加される．このため，変数消去で式の数が必ずしも単調に減らない．すなわち，最悪の場合には，収束しない可能性がある問題が一緒に明記されている．したがって，アルゴリズムの停止性を確保するために，ある固定の回数以上には，反復しない工夫がなされている．しかし，万一，そうした場合に陥ると，依存の可能性ありとせざるをえない．すなわち，完全には厳密ではなくなる．けれども，この論文中では，実際に PB を使って性能評価した結果，前処理段階も含めた総反復回数は，平均で 4.9〜5.7 回，最大で 17〜18 回であったことが示されている．すなわち，この程度の反復回数を上限とすれば，多くの場合厳密に解けるという主張である．これに対し，我々の提案手法は，シンプレックス法適用時には，まったく反復させない．その代わりに，全探索させて解空間を総当たりで調べる方式としている．それも，先に述べたように Branch and Bound 形式ではなく，ごく単純に整数格子点を巡るアルゴリズムとしている点でまったく異なる．

次に，シンプレックス法類以外の手法について，紙数の制約上必要と思われるものについてのみ簡単に触れる．

文献 17) では，厳密で高速な Delta test が提案されている．これは，本論文でも利用している分離テスト（Separability test）の考え方を，よりいっそう細かく分類し，さらに厳密に判定可能な場合を追加することで，より広範に適用可能とする．しかし，当然想定外の複雑な添え字式には厳密に解析できない．

文献 18) では，Λ test が提案されている．これは，多次元配列の添え字式を同時に満足する解を見いだすもので，特に組み添え字（coupled subscript）に対応することを目標にしている．この手法は多次元空間内

の超平面を幾何的に解析し，うまく射影することにより，強力でかつ高速に依存解析を行えるが，実数解を求めるにとどまっている．この文献では，整数計画法および線形計画法は，数百の制約式と変数を扱うためのもので，依存解析には複雑すぎると，最初に述べられている．しかし，本論文でその反証を示したい．

以上に対し，文献 4)，5) は，各種依存解析手法のサーベイ，そして比較論文となっている．前者では，PB，Linpack [19] 等，後者では，PB，LAPACK を用いて厳密性や解析時間を統計的に評価している．特に，文献 5) では，解析結果の有効性にまで踏み込んで，かつ厳密でない手法の代表として，Banerjee テストだけでなく，I-Test [20] も含めて詳細に調査されている．

4. Laputa テスト

ループ並列化時，一般に依存がなければ，そのまま並列実行が可能である．よって，依存解析問題では依存がないことを解析することが重視される．この観点から，文献 4)，5) の結論として，一般的なプログラムにおいて依存がないことを解析する能力は Banerjee テストと Omega テストでほぼ同じであること，科学技術計算プログラムにおいては Omega テストが Banerjee テストに比べ，約 1.7 倍優れていることが示されている．また，1 回の解析に要する平均時間の比較として，文献 4) では Omega テストが Banerjee テストより Linpack に対して約 63 倍，文献 5) では PB に対して約 22 倍，LAPACK に対して約 13 倍，時間がかかることが明らかにされている．よって，通常は Banerjee テストのみを行い，それで解決できなかった場合のみ Omega テストなどの厳密な依存解析手法を行えばよいといえる．しかし，Omega テストは解析速度が遅く，実装が困難であるため，実装が容易かつ解析速度の速い厳密な依存解析手法が求められる．

本論文で提案する Laputa テストはシンプレックス法[6] と全探索を核とするテストである．3 章で述べたとおり，同法を利用する場合でも，詳細に見れば，様々な方式が考えられる．ここで提案する手法では 2 章で述べたディオファンタス方程式を用いて，ループ上下限式から変数消去を行った後，同法を適用し☆，(実数) 解空間の頂点座標を求める．次に各頂点座標から探索空間を求め，探索空間内の整数格子点を全探索し，整数解を求める．なお，このとき得られる解空間は線形性の前提により，n 次元超空間において凸多面体の部分空間を形成する．

☆ ただし，3 章の類似研究に見られる反復解法ではない．

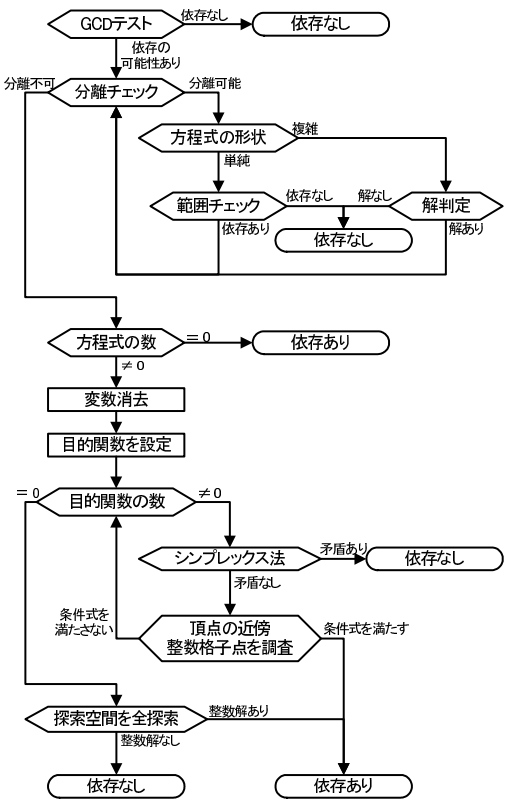

図 3 Laputa テストの流れ
Fig. 3 Flow of Laputa Test.

Laputa テストでは，依存がある場合には依存距離を求める．2.2 節で述べた通常の形ではなく，依存方向の情報を含め，$(i_1 - i_2)$ のとりうる範囲を返す．

Laputa テストでは，さらに解析時間を減らすために，できる限り全探索をせずに依存の有無を判定したい．この考察により，GCD テストと Banerjee テストと同等の処理を Laputa テストの前半の処理に組み込んだ．**図 3** はその Laputa テストの解析の流れの概略図である．以下，図 3 の流れに沿って Laputa テストのアルゴリズムを詳細に述べる．なお，ここでは 2.3 節で定式化した与えられた問題（P と記す）に現れる変数はすべてループの制御変数のみとする．ループの制御変数以外の外部変数が含まれる問題については，4.7 節で説明する．また，以下，例としては **図 4** の問題を用いて説明する．

4.1 GCD テスト

与えられた問題 P において，ディオファンタス方程式の変数の係数の GCD（Greatest Common Divisor）で右辺の定数項が割り切れるか否かにより解析を行う．割り切れなければ依存はない．割り切れれば

$$2 * i_1 - 2 * i_2 = 4 \quad (4)$$
$$j_1 + 3 * j_2 = 7 \quad (5)$$
$$13 * k_1 - 11 * l_2 = 19 \quad (6)$$
$$5 * k_1 + 3 * k_2 + 2 * l_1 = 23 \quad (7)$$
$$0 \leq i_1, i_2 \leq 10 \quad (8)$$
$$0 \leq j_1, j_2 \leq 10 \quad (9)$$
$$0 \leq k_1, k_2 \leq 5 \quad (10)$$
$$0 \leq l_1, l_2 \leq 5 \quad (11)$$

図 4 問題の例
Fig. 4 Example problem.

依存の可能性があるので，後の処理の簡単化のため，各係数と定数項をその GCD で割っておく．

図 4 の例では，式 (5)，(6)，(7) で係数の GCD は “1” となり，依存の可能性がある．式 (4) では，変数の係数の GCD は “2” となり，定数項の “4” が割り切れるため，やはり依存の可能性があることが分かる．以降，式 (4) を簡約した $i_1 - i_2 = 2$ (4') に置換する．

GCD テストは，実装の手間が少なく，解析時間が短いため，Laputa テストがそうであるように（図 3），他の依存解析手法のための “ふるい” として利用されることが多い．

4.2 分離チェック

任意のディオファンタス方程式と上下限式からなる系（与えられた問題 P）が以下の条件を満たすならば，その方程式と上下限式の組を他から分離して単独に部分問題として解析を行うことが可能である[1)]．

- 1 組の変数 V_1，V_2 のみで，ある 1 つの方程式が構成される．
- 変数 V_1，V_2 が他の方程式に含まれない．
- 変数 V（V_1，V_2 と区別する前の式の制御変数としての V；以下同様）の上下限値が整定数である．
- 変数 V 以外の変数の上下限式に V が含まれない．

複数の方程式と上下限式がこの条件を満たす場合は，それぞれ個別に解析する．個別に解析を行った結果，依存がない部分問題が 1 つでもあれば，入力された元問題 P においても依存がないことが分かる．分離チェックによってすべての方程式が分離し，すべてに依存があるならば，入力された元問題 P には依存があることが分かる．分離することができない方程式もしくは上下限式が残る場合（分離できなかった部分問題を P' と記す）には，複数の変数が互いに関係しあっているため，シンプレックス法と全探索を行う（4.3 節〜4.6 節）．

図 4 の例では式 (4)，(8) の組と式 (5)，(9) の組がそれぞれ分離可能であり，式 (6)，(7)，(10)，(11) は分離不可能で残る．

以下，分離できた部分問題に対する解析方法について詳細に述べる．

4.2.1 方程式の形状が単純な場合

方程式の係数がすべて “1” または “0” である場合を単純な形状の方程式と呼ぶことにする．さらに，単純な形状の方程式を以下の 3 種類に分類する．n は任意の整数である．

(1) $V_1 = n$ or $V_2 = n$
(2) $V_1 - V_2 = 0$
(3) $V_1 - V_2 = n$

これら単純な形状の方程式では変数 V の上下限値を用いて，簡単に依存の有無を厳密に判定することが可能である（図 3 で「範囲チェック」と記した処理）．

(1) $V_1 = n$ or $V_2 = n$ の場合
n が上下限の範囲内を満たせば依存があり，満たさなければ依存はない．
(2) $V_1 - V_2 = 0$ の場合
ループ独立依存がある．
(3) $V_1 - V_2 = n$ の場合
“(下限値 − 上限値) $\leq n \leq$ (上限値 − 下限値)” を満たせば依存があり，満たさなければ依存はない．

依存がある場合，提案手法では依存距離の可能な範囲を求める．(1) の場合はそれぞれ両辺に $-V_2$ を加えて，または V_1 から引いて，$V_1 - V_2$ の形にする．依存距離は “n − 上限値” から “n − 下限値”，または “下限値 − n” から “上限値 − n” の範囲といえる．(2) の場合，依存距離は “0”，(3) の場合，依存距離は “n” となる．

図 4 の例では，式 (4') $i_1 - i_2 = 2$ の形状が上記 (3) と一致する．定数項 “2” は式 (9) より，変数 i の (下限値−上限値) $= -10$ から (上限値−下限値) $= 10$ の範囲内の値である．よって，依存があること，さらに，依存距離は “2” であることが分かる．

4.2.2 方程式の形状が複雑な場合

図 3 で「解判定」と記した処理を以下のとおり行う．

$$a * V_1 + b * V_2 = n \quad (12)$$
（a，b，n は任意の整数；$a, b \neq 0$）

方程式が前項以外の一般式 (12) の場合を複雑な形状の方程式と呼ぶことにする．この方程式を用いて，変数 V_1，V_2 の上下限値の不等式 $l \leq V_1$，$V_2 \leq u$ から V_1 を消去する．V_1 を消去した後は式 (13)，(14) となる．

$$\left.\begin{array}{l}\lceil (n-a*u)/b \rceil \le V_2 \\ V_2 \le \lfloor (n-a*l)/b \rfloor \quad (a*b>0) \\ \lceil (n-a*l)/b \rceil \le V_2 \\ V_2 \le \lfloor (n-a*u)/b \rfloor \quad (a*b<0)\end{array}\right\} \quad (13)$$

$$l \le V_2 \le u \quad (14)$$

式 (13)，(14) を同時に満たす範囲を抽出し，$l' \le V_2 \le u'$ とする．このとき同時に満足する範囲が存在しない場合，元問題に矛盾があるため，依存はない矛盾のある問題は Banerjee テスト（inexact）でも依存がないことをチェックすることが可能である．次に l' から u' の間の整数値を順に式 (12) に代入し，V_1 の値を求める．求めた V_1 の値が 1 つでも整数値であれば依存がある．整数値にならなければ，依存はない．

依存があれば，依存距離を求めるため，式 (12) への代入を繰り返し，V_1 も整数値になる整数解の組をすべて求める．求めた整数解の組を $V_1 - V_2 = n$ に代入し，“n” の値の集合を求める．依存距離は “n” の最小値から最大値までの範囲として返す．

図 4 の例では，式 (5) が複雑な形状の方程式である．式 (5) を $j_1 = 7 - 3 * j_2$ と変形し，式 (9) に代入し，j_1 を消去すると式 (15)，(16) となる．

$$-1 \le j_2 \le \lfloor 7/3 \rfloor = 2 \quad (15)$$

$$0 \le j_2 \le 10 \quad (16)$$

式 (15)，(16) から重複部分を抽出すると，j_2 の範囲は $0 \le j_2 \le 2$ となり，とりうる整数値は $j_2 = 0, 1, 2$ となる．各値を式 (5) に代入し，j_1 の値を求めると $j_1 = 7, 4, 1$ となり，整数解を持つ．よって，依存があることが分かる．次に，式 (5) を満たす整数解の組 $(j_1, j_2) = (7, 0), (4, 1), (1, 2)$ の 3 組から j の依存距離を求める．$j_1 - j_2$ としては，それぞれ 7，3，-1 となり，依存距離のとりうる範囲は -1 から 7 を返す．

4.3 変数消去

分離できずに残った部分問題 P' がある場合，シンプレックス法と全探索を行う前に，残ったディオファンタス方程式を用いて以下の手順で変数消去を行う．これにより，方程式の数の分だけ次元を減らすことが可能である．

(1) 方程式 D1 の最左の変数 $v1$ を消去対象とする．
(2) 方程式 D1 を用いて，他の方程式から $v1$ を消去する．
(3) 方程式 D2〜Dm まで同じ処理を繰り返す．
（以上で，係数行列の最左に単位行列ができる）
(4) 方程式 D1〜Dm を用いて，上下限式から変数 $v1$〜vm を消去する．

分離チェック後，図 4 の問題は**図 5** となるので，まず，式 (17) の最左の変数 k_1 を消去対象とする．式 (17) を用いて，k_1 を消去すると式 (18) は式 (21) となる．

$$13 * k_1 - 11 * l_2 = 19 \quad (17)$$

$$5 * k_1 + 3 * k_2 + 2 * l_1 = 23 \quad (18)$$

$$0 \le k_{1,2} \le 5 \quad (19)$$

$$0 \le l_{1,2} \le 5 \quad (20)$$

図 5 分離チェック後，残る系（部分問題 P'）
Fig. 5 After separability check.

$$9 \le 26 * l_1 + 55 * l_2 \le 204 \quad (22)$$

$$0 \le l_1 \le 5 \quad (23)$$

$$0 \le 11 * l_2 \le 46 \quad (24)$$

図 6 変数消去後（部分問題 P''）
Fig. 6 After variable elimination.

$$39 * k_2 + 26 * l_1 + 55 * l_2 = 204 \quad (21)$$

次に式 (21) の最左の変数 k_2 を消去対象とする．式 (17) にはすでに k_2 が含まれないため，式 (21) を用いた消去は行わない．ディオファンタス方程式間での変数消去後，式 (17)，(21) を用いて，上下限式から変数 k_1，k_2 を消去する．変数消去後の部分問題 P' は等式を含まない形の**図 6** になる．

4.4 シンプレックス法と目的関数の設定

変数消去を行った後の上下限式を用いて，シンプレックス法を行う．今回は文献 21) のアルゴリズムをそのまま利用した．同法の詳細については，本論文の目的ではないので触れない．このアルゴリズムでは事前に実数解が存在するか否か，および解空間が有界か否かの判定，ならびに冗長な式の判定と削除を行う．実数解が存在しない場合，同法では条件式に矛盾があるという答えを返す．実数解が存在しなければ，整数解も存在しないため，依存がないことが分かる．実数解の存在判定は，Banerjee テスト（inexact）でも可能である．

シンプレックス法を適用するためには，与条件の係数行列とともに何らかの目的関数を設定する必要がある．このうち係数行列は変数消去後の上下限式そのものである．次に目的関数の設定方法であるが，本来線形計画法の目的は，与えられた等式，不等式が表す制約の下で，目的関数を最小もしくは，最大とする各変数の値を求めることである．ところが，Laputa テストでは，以下で述べるとおり，複数の目的関数を設定し，それぞれについて同法を適用することにより，同法の本来の目的とは異なるが，解空間の各頂点の座標を可能な限り求めることができる．今回の手法では，

目的関数として，上下限式に現れる変数の各変数軸に直交する超平面を表す関数を用いることを提案する．すなわち，変数消去後の上下限式に現れる変数が n 個の変数 $v1$〜vn の場合，目的関数は以下の式 (25) のすべて（$2n$ 種）が候補となる．

$$\begin{cases} Z = \pm v1 \\ Z = \pm v2 \\ \vdots \\ Z = \pm v(n-1) \\ Z = \pm vn \end{cases} \tag{25}$$

たとえば，$Z = +v1$ を目的関数とすることにより，上下限式すべての制約を満たす（実数）解空間内での $v1$ の最小値が求まり，$Z = -v1$ を目的関数とすることにより，$-v1$ の最小値，すなわち $v1$ の最大値が求まる．したがって，式 (25) の目的関数を順に適用することで，解空間内での各変数の値域を求めることが可能となる．この性質は後の全探索で非常に有用となる．

例の図 6 では係数行列として行列 (26) を得る．上下限式に含まれる変数は l_1 と l_2 であるので，目的関数は $Z = \pm l_1$ と $Z = \pm l_2$ を順に設定する．実際に係数行列とこれらの目的関数に同法を適用すると，4 つの頂点が以下のとおり求まる．

- $Z = l_1 \quad \rightarrow \ R1 = (0, 0.164)$
- $Z = -l_1 \ \rightarrow \ R2 = (5, 0)$
- $Z = l_2 \quad \rightarrow \ R3 = (0.346, 0)$
- $Z = -l_2 \ \rightarrow \ R4 = (0, 3.709)$

$$\begin{array}{c} \text{定数項} \quad l_1 \quad l_2 \\ \left(\begin{array}{cccc} 9 & 26 & 55 & \geq \\ 204 & 26 & 55 & \leq \\ 0 & 1 & 0 & \geq \\ 5 & 1 & 0 & \leq \\ 0 & 0 & 11 & \geq \\ 46 & 0 & 11 & \leq \end{array} \right) \end{array} \tag{26}$$

以上で，同法の適用は終わる．3 章の関連研究で見られるような同法自体の反復解法はとらない．

4.5 頂点の近傍整数格子点を調査

シンプレックス法によって求めた頂点の近傍整数格子点が解空間内の点であれば，整数解の候補となる．各目的関数によって求まった頂点 R の各座標値がすべて整数である場合は，この点 R そのものも整数解の候補となる．そこでまず，この点 R の座標を分離後の方程式に代入し，変数消去によって消去した変数の値をすべて求める．すべての消去した変数の値が整数値となるとき，整数解を持つので依存があることが分かる．整数値とならなかった場合は頂点 R の近傍整数格子点を同様に調査する．まず考える近傍整数格子点は頂点の座標の値をある 1 つの軸方向に ± 1 移動させた点とする．よって，n 次元の点であれば，2n 個の近傍整数格子点が存在する．しかし，たとえば目的関数 $Z = +v1$ によって求めた頂点座標の $v1$ 軸の値 $c1$ はすでにとりうる最小値となっているため，$v1$ の軸方向には -1 の移動はせず，$+1$ の移動のみでよい．近傍整数格子点に対してはまず，座標を変数消去後の上下限式に代入し，すべての式を満たすことを確認し，解空間内の点であることを確認する．解空間内の点であれば，部分問題 P' の方程式に代入し，消去した変数の値をすべて求め，整数解を持てば依存があることが分かる．

また，頂点の各座標に小数値が含まれる場合は整数値に丸める必要がある．このとき，切り上げるのか，切り捨てるのかを適切に判断しなければならない．もし，目的関数と同じ軸方向の値の場合は，上と同様にすでに最大値か最小値かにより，切上げ/切捨てをただちに適切に判断することが可能である．その他の軸方向の値はすべての頂点を求めなければ，判断できない．そのため，試作版の Laputa テストでは小数点以下を四捨五入し，丸めた頂点を点 R' とする．すると点 R' も解空間内の点であることが保証されないため，まず変数消去後の上下限式に代入し，すべての式を満たす（解空間内である）ことを確認した後，先と同様に処理する．もし点 R' が整数解とならなかった場合は，さらに範囲を広げて点 R' の近傍格子点を調査する．

依存があることが分かった場合は，依存距離を求める．ここでは，シンプレックス法を利用して求めることを提案する．具体的にはまず，分離チェック後の部分問題 P' を用いて，係数行列を作成する．P' に元の制御変数 $V1, V2, \ldots, Vn$ が含まれるならば，目的関数を $Z = \pm(V1_1 - V1_2) \ldots Z = \pm(Vn_1 - Vn_2)$ と設定する．たとえば目的関数 $Z = \pm(V1_1 - V1_2)$ によって求まる値の範囲は $V1$ の依存距離の可能な範囲そのものである．

解空間の頂点も近傍整数格子点も整数解とならなければ，目的関数を変更し，他の頂点を求め，先と同様の処理を行う．

図 6 の上下限式と各頂点を図に表すと**図 7** となる．図中の灰色のハッチング部分が解空間であり，白丸が頂点 $R1$〜$R4$ である．$R1 = (0, 0.164)$ は目的関数 $Z = l_1$ によって求めた点であるので，0 が l_1 軸の最小値である．0.164 を四捨五入すると 0 となり，

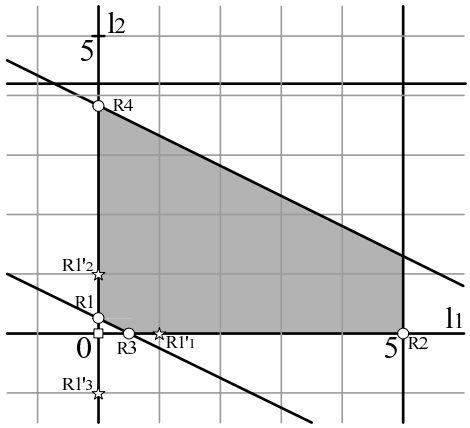

図 7 近傍整数格子点の調査
Fig. 7 Check from vertex R1.

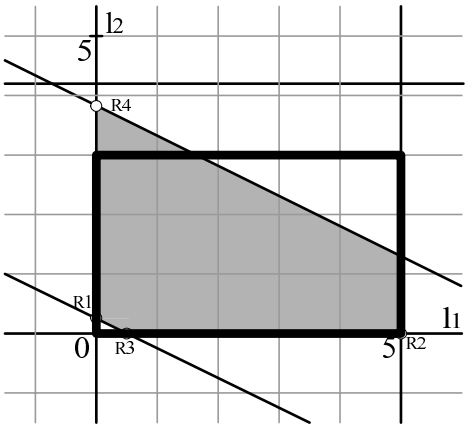

図 8 探索領域
Fig. 8 Search area.

$R1' = (0,0)$ が得られる．同図中の白い四角が $R1'$ である．$R1'$ の座標は上下限式 (22)〜(24) を満たさない（図の灰色のハッチング外）ので，求める整数解ではない．次に近傍整数格子点を求める．近傍整数格子点は l_1 軸に沿って $+1$ 移動した $R1'_1 = (1,0)$，l_2 軸に沿って ± 1 移動した $R1'_2 = (0,1)$，$R1'_3 = (0,-1)$ の 3 点である．図中の白い星印が $R1'_1$〜$R1'_3$ である．$R1'_1$，$R1'_2$ は上下限式を満たすが，$R1'_3$ は満たさないので，$R1'_1$，$R1'_2$ をそれぞれ部分問題 P' の方程式 (17)，(18) に代入し，k_1，k_2 の値を求める．$R1'_1$，$R1'_2$ ともに k_1，k_2 の値は小数値になるため，部分問題の整数解ではない．続いて，$R2$，$R3$ に対しても同様の処理を行うが，整数解は存在しない．$R4$ の座標の l_2 軸の値を四捨五入した $R4' = (0,3)$ は上下限式を満たし，$k_1 = 4$，$k_2 = 1$ と整数値となる．よって，$(k_1, k_2, l_1, l_2) = (4,1,0,3)$ が整数解となり，図 4 の元問題には依存があることが分かる．

依存があることが分かったので，次に k と l に対する依存距離を求める．図 5 の問題を用いて係数行列を作成する．目的関数は $Z = \pm(k_1 - k_2)$ と $Z = \pm(l_1 - l_2)$ とする．シンプレックス法を適用すると，$-3.54 \leq k_1 - k_2 \leq 4.6$，$-3.71 \leq l_1 - l_2 \leq 5$ となり，k の依存距離は -3〜4，l の依存距離は -3〜5 の範囲であることが分かる．

4.6 全 探 索

前節で述べた解空間の各頂点とその近傍整数格子点に解が存在しない場合は全探索を行う．全探索を行う範囲は先の手順で得られている各頂点の座標から次のように求める．得られている解空間の各頂点の座標から座標軸ごとの最小値と最大値を求める．整数値でない最小値は切り上げて，最大値は切り捨てて整数値にする．つまり，各座標軸の最小値と最大値で囲まれる超直方体を探索空間とする．この探索空間を全探索することにより，解空間を網羅することが可能である．探索点が整数解か否かを判定する方法は前節の頂点の近傍整数格子点を調査した手法と同じである．試作版の Laputa テストでは探索空間を全探索する最も単純な方法として，n 次元空間の場合，1 次元目から n 次元目まで順に最小値から最大値までを網羅するように探索させている．しかし，探索空間は解空間を十分に包み込む形になるため，探索空間の頂点付近は解空間でない部分が多い．よって，早期に整数解を発見するためには，たとえば解空間の座標から重心を求め，重心から各座標軸方向に広がるように探索を行うことも考えられる．

依存距離の求め方は前節の「頂点の近傍整数格子点を調査」の方法と同様に，分離後の部分問題 P' に対してシンプレックス法を用いて求める．

図 6 の問題は近傍整数格子点が整数解となるため，実際には全探索は行われないが，探索空間の決定の方法を説明するために，図 6 の問題を用いて以下説明する．シンプレックス法により，解空間の頂点座標は先の $R1$〜$R4$ の 4 点が求まる．4 つの頂点座標から l_1 の範囲が $0 \leq l_1 \leq 5$，l_2 の範囲が $0 \leq l_2 \leq 3$ の探索空間が求まる．すなわち，**図 8** の太い線で囲まれた領域が探索空間である．

4.7 問題に外部変数が含まれる場合への対応

依存解析対象のループと，ネストレベルでより深いループの制御変数以外の変数を “外部変数” と呼ぶ．ループ制御変数はループの開始値と終端値から静的に値の範囲が決まる．しかし，外部変数には静的に値が求まる変数と，動的に値が求まる変数がある．静的に

値が求まる変数は，定数伝搬[1] などの最適化を依存解析の前に行うことにより，変数を定数に置き換えることが可能である．けれども，動的に値が求まる変数が問題に含まれると，厳密な依存解析は一般には不可能である．

Omega テストは問題に外部変数が含まれる場合，条件付きで依存があると結果を返す．条件付きの結果を返すことにより，コンパイラは条件を満たさない場合に並列実行するようにコード生成ができる．試作版の Laputa テストでは問題に外部変数が含まれると，依存の可能性があることを返す．Laputa テストのアルゴリズムでは外部変数を含むすべてのケースに対応することは難しいが，以下で述べる特定の条件を満たす場合のみ，条件付きで依存の有無を解析することが可能である．

(1) 外部変数が変数消去（4.3 節）によって消去できる場合
方程式と上下限式に同じ外部変数が含まれる場合，変数消去を行い上下限式から外部変数を消去することが可能である．後は 4 章のアルゴリズムでシンプレックス法を行い，解空間を求める．解空間内の整数格子点があれば，その座標を方程式に代入し，外部変数のとりうる値の集合を求めることができる．実行時に外部変数がこの集合の中の値でなければ依存はないといえる．

(2) 変数消去後，方程式のみに外部変数が残った場合
外部変数が 1 つの場合（O とする）は，目的関数を “$Z = \pm O$” とする．後は，4 章の通常どおりの処理で，シンプレックス法を適用すると，依存があるときの外部変数の範囲が求まる．また，複数の外部変数 $o_1, o_2 \ldots o_k$ が残っている場合は，方程式中の $\sum_{i=1}^{k} a_i * O_i$ を新たに O と置き換えて，上記に帰着させる．

(3) 上下限式の上下いずれかに外部変数が残った場合
外部変数を含む v の上（下）限式を除いた問題を用いて係数行列を作成する．目的関数に各変数軸に直交する関数を設定し，シンプレックス法を適用すると，v のとりうる最小値，もしくは最大値が求まる．最小値より外部変数の値が小さい，もしくは最大値より外部変数の値が大きいときは依存はない．また，ディオファンタス方程式にも外部変数が含まれる場合は (2) と同様に目的関数を生成し，外部変数の範囲を求める．

(4) 上下限式両方に外部変数が含まれる場合
上限式，下限式ともに外部変数が含まれる制御変数は，範囲が静的に解析できないため，外部変数の 1 種と見なして扱う．その後，上の (1)〜(3) に帰着できれば，対応する処理を行う．

表 1 実験環境
Table 1 Experimental environment.

計算機	PC-AT 互換機 CPU Celeron 700 MHz 主記憶 128 MB
OS	RedHat Linux Kernel 2.4.20

5. 評　価

本章では，**付録**の 55 種類のテスト用の問題を考案し，それらを用いて，試作版 Laputa テストと Omega テストの解析時間を比較した．また，実プログラムに対する評価のために Linpack と Perfect Club Benchmarks に含まれる 13 種類のプログラムに対する解析時間も比較した．実験は**表 1** の環境で行った．解析時間の測定には RDTSC（read time stamp counter）命令[22] を用いた．RDTSC 命令は PC が起動してからのクロックサイクル数を取得する命令である．解析の前後に RDTSC 命令を挿入し解析にかかったクロックサイクル数を求めた．クロックサイクル数を CPU のクロック数で割り，解析時間を求めた．しかし，両テストともに解析時間が測定のたびに微妙にばらついたため，5〜10 回の試行をしたうえで，それらの解析時間の平均を求めた．ただし，Omega テストは 1 回目の解析時間が 2 回目以降と比べて，10 倍以上長かったため，2 回目以降の平均を求めた．今回，実験に用いた Omega テストは MARYLAND 大学で開発された Petit ツール V1.2[23] を用いた．Petit ツール内の Omega テストの処理の前後に RDTSC 命令を挿入し，解析時間の測定を行った．Petit ツールに入力するため，問題のディオファンタス方程式と上下限式を含む FORTRAN のプログラムを作成した．

実験に使用したテスト用の問題は，試作版 Laputa テストの内部の各処理をテストすることを目的として，最悪の場合も含め，すべての場合分けを網羅させるために，作為的な例になっても含めるように考慮した．それゆえ，実際にはまったく意味のないプログラムも含まれる（特に，下の (6)，(7) 中）．評価を行うにあたり，試作版 Laputa テストの流れに沿って，考案した問題群を次の 7 種類に大分類した．

(1) GCD テストで依存がないと判定できるクラス
(2) 分離した問題に依存がないと判定できるクラス

(3) 分離した問題すべてに依存があると判定できるクラス
(4) シンプレックス法で矛盾が生じるクラス
(5) 頂点の近傍整数格子点が整数解となるクラス
(6) 全探索の結果，整数解があると判定できるクラス
(7) 全探索の結果，整数解がないと判定できるクラス

上記は，依存解析問題全般について網羅的に調査するために分類したものであるが，Laputa テストの核であるシンプレックス法と全探索を組み合わせた部分に着目するには，このうち (5)〜(7) が重要である．したがって，後の 6 章では，これら (5)〜(7) を中心に考察する．テスト用の各問題に対する解析時間を**表 2** にまとめた．表の “No.” 欄が付録の問題番号である．全探索を行う (6)，(7) に含まれる問題に対しては探索回数を求め，付録の問題名の下の括弧内に記した．また，Laputa テストと Omega テストの依存解析結果はすべて同じ結果となった．このことから，テスト問題に対する解析性能は Omega テストとほぼ同等であることが確かめられた．解析時間については章を改め，詳細に述べる．

実プログラムとして，Linpack と PB について調査した．まず，現れたすべての問題について，大分類 (1)〜(7) に分類し，各事例を数え上げた．次に上に述べた理由から，大分類 (5)〜(7) に着目し，Laputa テストと Omega テストによる平均解析時間を実際に求めてみた．これらの結果を**表 3** にあげる．これらの評価を行う際，対象は多重ループ内の問題に限定し，ループの初期値・終値に含まれる外部変数が，静的に解析不可能な場合は定数（初期値が不明な場合は 0，終値が不明な場合は 100）に置き換えた．この表 3 から，「依存あり」となった問題で，単純な与式である大分類 (3) 以外の複雑な場合には，すべて大分類 (5) であった．したがって，一般にも「依存あり」となる問題は，この大分類 (5) である確率がきわめて高いと考える．換言すれば，ほとんどの場合シンプレックス法で求めた解空間の頂点近傍に整数解が見つかるので，網羅的に全探索する必要がないことが分かる．

6. 考　　察

表 2 の大分類 (1)〜(4) の結果から，Laputa テストは GCD テストと Banerjee テストを内包することにより，簡単な依存解析問題に対して，厳密でかつ高速に依存解析ができたことが分かる．これらのクラスでは，Omega テストは少なくとも 1 ms 以上は解析に

表 2 テスト用の問題に対する測定結果
Table 2 Timing result of a test problem.

	No.	Laputa[ms]	Omega[ms]
(1) GCD テスト 依存なし	101	0.006	1.670
	102	0.007	2.018
	103	0.006	2.058
	104	0.006	4.002
	105	0.006	1.719
(2) 分離後 依存なし	201	0.028	1.509
	202	0.026	3.052
	203	0.030	1.618
	204	0.034	3.969
	205	0.030	4.738
	206	0.030	7.137
	207	0.044	1.691
	208	0.046	1.731
	209	0.052	2.063
	210	0.050	1.986
	211	0.049	2.440
(3) 分離後 依存あり	301	0.030	5.642
	302	0.031	9.474
	303	0.034	3.089
	304	0.051	6.632
	305	0.034	17.851
	306	0.035	19.743
	307	0.037	23.324
	308	0.037	19.234
	309	0.039	3.738
	310	0.035	2.458
	311	0.057	2.534
	312	0.061	12.613
	313	0.066	49.800
(4) シンプレックス法 矛盾あり	401	0.138	2.155
	402	0.288	7.614
	403	0.187	2.495
	404	0.138	4.086
	405	0.179	2.513
(5) 近傍整数格子点 依存あり	501	0.192	8.239
	502	0.191	3.976
	503	0.172	13.684
	504	0.203	56.789
	505	0.165	38.834
	506	0.510	5.358
	507	0.185	14.840
(6) 全探索 依存あり	601	0.459	10.446
	602	0.410	23.452
	603	0.904	23.644
	604	724.120	16.266
	605	0.484	28.638
	606	50.215	48.411
	607	0.552	5.646
	608	0.509	7.468
(7) 全探索 依存なし	701	0.295	2.291
	702	1.199	2.453
	703	0.389	2.127
	704	2.339	2.712
	705	42.630	3.130
	706	441.087	33.509

表 3 実プログラムに対する測定結果
Table 3 Timing result of a real program.

	分類	問題数	Laputa [ms]	Omega [ms]
Linpack	総数	8,156	-	-
	(1)〜(4)	7,594	-	-
	(5)	550	0.182	7.189
	(6)	0	-	-
	(7)	12	0.182	1.896
Perfect Benchmark	総数	19,459	-	-
	(1)〜(4)	19,330	-	-
	(5)	129	0.277	16.797
	(6)	0	-	-
	(7)	0	-	-

時間がかかっている．特に $V_{1or2} = n$ の形のディオファンタス方程式を含む問題に対しては異常に遅くなることも分かった．したがって，Omega テストを使う場合にその前処理として GCD テストと Banerjee テストを行うべきである．

次に，Laputa テストの核であるシンプレックス法と全探索の部分を評価するために，表 2，表 3 の大分類 (5)〜(7) の結果について考察する．

表 2 の大分類 (5) の結果から，解空間の近傍整数格子点が整数解である場合，Laputa テストは Omega テストに比べて，約 10 倍から 280 倍高速に解析が可能であることが分かる．また，表 2 で大分類 (6) に含まれる問題の中にも，近傍整数格子点の座標をさらに ±1 した格子点が解である場合も多かった．すなわち，試作版の近傍整数格子点の探索を，今の ±1 から ±2 に広げるだけで，問題によっては大分類 (6) から大分類 (5) に変わり，全探索を行う確率をさらに減らすことができると考える．

表 2 の大分類 (6) の No.606 の結果から，全探索の結果，依存がある場合は 5,000 回程度の探索回数で Omega テストと同等の解析時間となることが分かる．しかし，探索回数が 1,000 回を超えるような問題は，表 3 に示したとおり，Linpack，PB には存在せず，No.604，606 のように，解空間のきわめて限られた点でのみ整数解を持つように作為的に作成しないかぎり，実際にはほとんど存在しないと考えられる．よって，この大分類 (6) の場合においても通常は Laputa テストが Omega テストより高速であると考える．

表 2 の大分類 (7) の結果から，Omega テストについては依存がある場合に比べて依存がない場合の方が高速に解析が可能であることが分かる．Laputa テストはこの場合 200 回程度の探索回数で Omega テストと同等の解析時間となることが分かる．解空間は一般に次元数が多い場合，つまり，配列の添え字に使われる変数（ループネスト）が多い場合に，大きくなりやすい．実際のプログラムにおいて，解空間の次元が多くなる問題がどの程度存在するかを調べるために，Linpack の内部で利用されるすべての配列に対して，次元数と添え字式に表れる制御変数の数を調査した．調査の結果，添え字には最大 2 個の制御変数しか表れないことが分かった．よって，依存がある場合と同じように，実際には依存がないにもかかわらず，解空間が大きい問題は一般にも少ないと予想している．

次に表 3 から，Linpack，PB への適用結果について，さらなる考察を加える．表 3 で大分類 (6)，(7) に含まれるのは Linpack の 12 個の問題のみであった．しかも，これら 12 例すべてにおいて，総探索回数は 1 回だけで終了した．また，これらすべてにおいて，探索空間は 1 点に縮退しており，かつ，整数解ではなかった．これらの 12 例に対する Laputa テストの解析速度は，Omega テストの約 10 倍であった．Linpack，PB 全体で大分類 (5) 含まれる問題に対しての Laputa テストの解析速度は，同じく約 40 倍〜約 60 倍高速であることが分かった．以上の結果から，Laputa テストの核であるシンプレックス法と全探索の部分について，実プログラム Linpack，PB に対して Omega テストより高速であると結論づけることができると考える．

また，大分類 (6)，(7) の結果から，Laputa テスト全体で見ると，やはり時間がかかるのは，全探索であることが分かる．よって，全探索の方法を改善することにより，さらに高速化が図れると考える．

全探索の方法の改善に関しては，4.6 節でも述べたが，探索空間の頂点付近は解空間ではないことが多い（たとえば図 8 の太枠の長方形の右上の白い三角部分）ため，解空間の頂点座標から解空間の重心を求め，重心から各座標軸方向に広がるように探索をした方が早期に整数解を発見する可能性が高くなる．しかし，依存がない場合は，結局探索空間を全探索しなければならないため，重心から探索しても解析時間は変わらない．依存がない場合に解析時間を減らすためには，探索空間内の解空間ではない領域の探索回数を減らさなければならない．この改良した探索方法は紙面の都合もあるので，別論文でまとめる予定である．

7. おわりに

新しい厳密な配列要素間のデータ依存解析手法である Laputa テストを提案し，アルゴリズムと実現手法について述べた．アルゴリズムは一般的に使われているシンプレックス法と全探索に，GCD テスト，Banerjee テストを簡単な場合分けで統合的に組み合わせて

おり，全体として比較的簡単に実装が可能なテストである．また，性能評価の結果，一般に厳密な解析をしつつ，Laputa テストは多くの場合，Omega テストよりも高速に依存解析が行えることが分かった．しかしながら，きわめて稀なケースではあるが，探索空間が広く，全探索に要する時間が長くなる場合がある．今後の課題として，そうした場合に対応できる全探索の方法の改善が必要である．

謝辞 本研究は一部科学研究費（No.13480085）による．また，本研究は総務省戦略的情報通信研究開発制度の一環として行われたものである．

参 考 文 献

1) Zima, H. and Chapman, B.: *Supercompilers for Parallel and Vector Computers*, 1st edition Frontier, ACM Press, New York, (1990).
2) Banerjee, U.: *Dependence Analysis*, Loop Transformations for Restructuring Compilers, 1st edition, Kluwer Academic, Norwell, Massachusetts (1997).
3) Pugh, W. and Wonnacott, D.: Eliminating False Data Dependences using the Omega Test, *Proc. ACM SIGPLAN 1992 Conference on Programming language design and implementation*, *SIGPLAN, ACM*, San Francisco, California, pp.140–151, ACM Press (1992).
4) Psarris, K. and Kyriakopoulos, K.: Data Dependence Testing in Practice, *1999 International Conference on Parallel Architectures and Compilation Techniques*, Newport Beach, California, pp.264–273, IEEE (1999).
5) Psarris, K. and Kyriakopoulos, K.: The Impact of Data Dependence Analysis on Compilation and Program Parallelization, *Proc. 17th annual international conference on Supercomputing 2003*, San Francisco, CA, pp.205–214 (2003).
6) 小野勝章：計算を中心とした線形計画法，日科技連 (1996).
7) Maydan, D.E., Hennessy, J.L. and Lam, M.S.: Efficient and Exact Data Dependence Analysis, *Proc. ACM SIGPLAN 1991 Conference on Programming Language Design and Implementation*, Toronto, Ontario, Canada, pp.1–14 (1991).
8) Banerjee, U.: *Dependence Analysis for Supercomputing*, Kluwer Academic (1988).
9) Darttzig, G. and Eaves, B.C.: Fourier-motzkin elimination and its dual, *Journal of Combinatorial Theory*, A(14), pp.288–297 (1973).
10) Eigenmann, R., Hoeflinger, J., Li, Z. and Padua, D.: Experience in the Automatic Parallelization of Four Perfect Benchmark Programs, *Languages and Compilers for Parallel Computing, 4th International Workshop*, Banerjee, U., Gelernter, D., Nicolau, A. and Padua, D. (Eds.), Santa Clara, CA, Springer-Verlag (1991).
11) Berry, M., et al.: The Perfect Club Benchmarks: Effective Performance Evaluation of Supercomputers, *Center for Supercomputing Research and Development, University of Illinois at Urbana-Champaign, CSRD report #827* (1989).
12) Petersen, P.M. and Padua, D.A.: Static and Dynamic Evaluation of Data Dependence Analysis Techniques, *IEEE Trans. Parallel and Distributed Systems*, Vol.7, No.11, pp.1121–1132 (1996).
13) http://www.netlib.org/lapack/
14) http://www.netlib.org/eispack/
15) Eisenbeis, C. and Sogno, J.C.: A General Algorithm for Data Dependence Analysis, *Proc. 6th ACM International Conference on Supercomputing*, July 1992, Washington, DC, USA. pp.292–302, ACM (1992).
16) Wallace, D.R.: Dependence of Multi-Dimensional Array References, *Proc. 2nd International Conference on Supercomputing*, pp.418–428 (1988).
17) Goff, G., Kennedy, K. and Tseng, C.W.: Practical Dependence Testing, *Proc. ACM SIGPLAN 1991 Conference on Programming Language Design and Implementation*, Toronto, Ontario, Canada, pp.15–29 (1991).
18) Li, Z., Yew, P.-C. and Zhu, C.-Q.: An Efficient Data Dependence Analysis for Parallelizing Compilers, *IEEE Trans. Parallel and Distributed Systems*, Vol.1, No.1, pp.26–34 (1990).
19) http://www.netlib.org/linpack/
20) Kong, X., Klappholz, D. and Psarris, K.: The I-Test: An Improved Dependence Test for Automatic Parallelization and Vectorization, *IEEE Trans. Parallel and Distributed Systems*, Vol.2, No.3, pp.342–349 (1991).
21) 奥村晴彦：Java によるアルゴリズム事典，技術評論社 (2003).
22) Using the RDTSC Instruction for Performance Monitoring. http://cedar.intel.com/software/idap/media/pd/rdtscpm1.pdf
23) Frameworks and Algorithms for the Analysis and Transformation of Scientific Programs. http://www.cs.umd.edu/projects/omega/

付録 テスト用に考案した問題群

ここでは，紙数の制限から上下限式は i_1，i_2 等を区別せずに書き下す．

A.1 GCD テスト：依存なし

$[No.101]$ $2*i_1-4*i_2=3,\quad 0\le i\le 10$

$[No.102]$ $2*i_1-4*i_2=6,\ \ 2*j_1-4*j_2=3,\ \ 0\le i,\ j\le 10$

$[No.103]$ $2*i_1-4*i_2=3,\ \ 2*j_1-4*j_2=6,\ \ 0\le i,\ j\le 10$

$[No.104]$ $20*i_1+30*j_2=5,\quad 0\le i,\ j\le 10$

$[No.105]$ $179*i_1-358*i_2=190,\quad 0\le i\le 10$

A.2 分離後：依存なし

$[No.201]$ $i_1=5,\quad 0\le i\le 4$

$[No.202]$ $i_2=10,\quad 0\le i\le 5$

$[No.203]$ $i_1-i_2=20,\quad 0\le i\le 10$

$[No.204]$ $i_1=5,\ \ j_2=10,\ \ 0\le i\le 10,\ \ 0\le j\le 5$

$[No.205]$ $i_1-j_2=5,\ \ k_1=5,\ \ 0\le i,\ j\le 10,\ \ 0\le k\le 3$

$[No.206]$ $i_1-j_2=10,\ \ k_2=20,\ \ 0\le i,\ j,\ k\le 10$

$[No.207]$ $i_1-2*i_2=30,\quad 0\le i\le 10$

$[No.208]$ $2*i_1-5*i_2=3,\quad 0\le i\le 3$

$[No.209]$ $i_1=5,\ \ 2*j_1-5*j_2=3,\ \ 0\le i\le 10,\ \ 0\le j\le 3$

$[No.210]$ $i_1-i_2=0,\ \ j_1-2*j_2=30,\ \ 0\le i,\ j\le 10$

$[No.211]$ $i_1-j_2=5,\ \ i_2+5*j_1=10,\ \ k_1-2*k_2=30,\ \ 0\le i,\ j,\ k\le 10$

A.3 分離後：依存あり

$[No.301]$ $i_1=5,\quad 0\le i\le 10$

$[No.302]$ $i_2=5,\quad 0\le i\le 10$

$[No.303]$ $i_1-i_2=5,\quad 0\le i\le 10$

$[No.304]$ $2*i_1-i_2=3,\quad 0\le i\le 10$

$[No.305]$ $i_1=5,\ \ j_1=5,\ \ 0\le i,\ j\le 10$

$[No.306]$ $i_2=5,\ \ j_2=5,\ \ 0\le i,\ j\le 10$

$[No.307]$ $i_1=5,\ \ j_2=5,\ \ 0\le i,\ j\le 10$

$[No.308]$ $i_2=5,\ \ j_1=5,\ \ 0\le i,\ j\le 10$

$[No.309]$ $i_1-i_2=5,\ \ j_1-j_2=5,\ \ 0\le i,\ j\le 10$

$[No.310]$ $i_1-i_2=0,\ \ j_1-j_2=0,\ \ 0\le i,\ j\le 10$

$[No.311]$ $i_1-i_2=0,\ \ j_1+j_2=0,\ \ 0\le i,\ j\le 10$

$[No.312]$ $2*i_1-i_2=3,\ \ j_1-2*j_2=3,\ \ 0\le i,\ j\le 10$

$[No.313]$ $2*i_1-i_2=3,\ \ j_1=5,\ \ k_2=10,\ \ 0\le i,\ j,\ k\le 10$

A.4 シンプレックス法：矛盾あり

$[No.401]$ $3*i_1-2*j_2=100,\ \ 2*i_2-4*j_1=80,\ \ 0\le i,\ j\le 10$

$[No.402]$ $i_1-i_2=0,\ \ i_1-k_2=0,\ \ j_1-j_2=0,\ \ 0\le i\le 100,\ \ i+1\le j,\ \ k\le 100$

$[No.403]$ $i_1-i_2=0,\ \ 3*j_1-2*k_2=100,\ \ 2*j_2-4*k_1=80,\ \ 0\le i,\ j,\ k\le 10$

$[No.404]$ $3*i_1-2*j_2=100,\ \ 0\le i,\ j\le 10$

$[No.405]$ $2*i_1-i_2=3,\ \ 3*j_1-2*k_2=5,\ \ 2*j_2-4*k_1=80,\ \ 0\le i,\ j,\ k\le 10$

A.5 近傍整数格子点：依存あり

$[No.501]$ $5*i_1-3*i_2+4*j_1-6*j_2=10,\ \ 3*i_1+2*i_2-4*j_1-5*j_2=6,\ \ 0\le i,\ j\le 10$

$[No.502]$ $i_1-i_2=10,\ \ j_1-j_2=6,\ \ 0\le i\le 20,\ \ i\le j\le 20$

$[No.503]$ $i_1-j_2=10,\ \ i_2-j_1=6,\ \ 0\le i,\ j\le 20$

$[No.504]$ $3*i_1-6*j_2=6,\ \ i_2-j_1=6,\ \ k_1-k_2=3,\ \ 0\le i,\ j,\ k\le 20$

$[No.505]$ $i_1-j_2=3,\ \ 0\le i,\ j\le 10$

$[No.506]$ $2*i_1-2*i_2=4,\ \ j_1+3*j_2=7,\ \ 13*k_1-11*l_2=19,\ \ 5*k_1+3*k_2+2*l_1=23,\ \ 0\le i,\ j\le 10,\ \ 0\le k,\ l\le 5$

$[No.507]$ $23*i_1-13*j_2=10,\ \ i_2+j_1=10,\ \ -20\le i\le 20,\ \ i-20\le j\le 20$

A.6 全探索：依存あり

$[No.601]$ (15) $13*i_1-11*j_2=19,\ \ 3*i_2+2*j_1=23,\ \ 0\le i,\ j\le 10$

$[No.602]$ (4) $13*i_1-11*j_2=19,\ \ 0\le i,\ j\le 10$

$[No.603]$ (128) $179*i_1-j_2=589,\ \ 1\le i\le 10,\ \ 0\le j\le 200$

$[No.604]$ (80930) $13*i_1-11*j_2=19,\ \ 179*j_1-k_2=589,\ \ 0\le i\le 10,\ \ 1\le j\le 10,\ \ 0\le k\le 200$

$[No.605]$ (4) $13*i_1-11*j_2=19,\ \ 75*k_1-k_2=0,\ \ 0\le i,\ j\le 10,\ \ 1\le k\le 100$

$$[No.606]\ (5660)\quad \begin{cases} 37*i_1-31*i_2=6 \\ 37*i_1-29*k_2=8 \\ 47*j_1-43*j_2=4 \\ -50\le i\le 100,\ i-51\le j,\ k\le 100 \end{cases}$$

$$[No.607]\ (37)\quad \begin{cases} 79*i_1-19*j_2=3 \\ 23*j_1-47*i_2=-1 \\ -10\le i\le 15,\ i-10\le j\le 15 \end{cases}$$

$$[No.608]\ (23)\quad \begin{cases} 49*i_1-47*j_2=2,\ 17*j_1+13*i_2=4 \\ -20\le i\le 20,\ i-20\le j\le 20 \end{cases}$$

A.7 全探索：依存なし

$$[No.701]\ (20)\quad \begin{cases} i_1-i_2+10*j_1-10*j_2=5 \\ i_1-j_2=-2 \\ 0\le i\le 4,\ \ i\le j\le 9 \end{cases}$$

$$[No.702]\ (252)\quad \begin{cases} 79*i_1-7*j_2=2 \\ 29*i_2-49*j_1=10 \\ -20\le i\le 20,\ i-20\le j\le 20 \end{cases}$$

$$[No.703]\ (36)\quad \begin{cases} 13*i_1-7*j_2=7 \\ -5*i_2+11*j_1=43 \\ 0\le i,\ j\le 8 \end{cases}$$

$$[No.704]\ (243)\quad \begin{cases} 13*i_1-5*j_2=5,\ 11*j_1-5*k_2=27 \\ 7*k_1-3*i_2=38 \\ 0\le i,\ j,\ k\le 8 \end{cases}$$

$$[No.705]\ (2916)\quad \begin{cases} 23*i_1-7*j_2=7,\ 17*j_1-7*k_2=5 \\ 13*k_1-5*l_2=7,\ 11*l_1-5*i_2=5 \\ 0\le i,\ j,\ k,\ l\le 8 \end{cases}$$

$$[No.706]\ (19683)\quad \begin{cases} 13*i_1-5*j_2=7,\ 23*j_1-7*k_2=7 \\ 11*k_1-5*l_2=5,\ 17*l_1-7*m_2=5 \\ 29*m_1-11*i_2=8 \\ 0\le i,\ j,\ k,\ l\le 8 \end{cases}$$

(平成 16 年 5 月 19 日受付)
(平成 16 年 9 月 29 日採録)

峰尾　昌明（学生会員）

2000 年和歌山大学システム工学部情報通信システム学科卒業．2002 年同大学大学院博士前期課程修了．現在同大学院博士後期課程在学中．自動並列化コンパイラ，分散並列処理，インターネット等の研究に従事．

上原哲太郎（正会員）

1990 年京都大学工学部情報工学科卒業．1992 年同大学大学院修士課程修了．1995 年同大学院博士後期課程研究指導認定退学．同年同大学院工学研究科助手．1996 年和歌山大学情報処理センター講師．1997 年同大学システム情報学センター講師．2000 年同大学システム工学部情報通信システム学科講師．2003 年京都大学工学研究科附属情報センター助教授，現在に至る．自動並列化コンパイラ，分散並列処理，システム運用技術，インターネットセキュリティ等の研究に従事．京都大学博士（工学），日本ソフトウェア科学会，CIEC 各会員．

齋藤　彰一（正会員）

1993 年立命館大学理工学部情報工学科卒業．1995 年同大学大学院博士前期課程修了．1998 年同大学院博士後期課程単位習得満期退学．同年和歌山大学システム工学部情報通信システム学科助手．2003 年同講師，現在に至る．オペレーティングシステム，分散並列処理，インターネット等の研究に従事．博士（工学），日本ソフトウェア科学会，ACM，IEEE-CS 各会員．

國枝　義敏（正会員）

1980 年京都大学工学部情報工学科卒業．1982 年同大学大学院修士課程修了．同年京都大学工学部情報工学科助手．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．2004 年立命館大学情報理工学部情報システム学科教授，現在に至る．工学博士．主として，計算機ソフトウェア，システムプログラム，言語処理系，超高速計算等の分野に関する研究に従事．電子情報通信学会，ACM，IEEE-CS 各会員．