2004－MPS－51　(3)
2004／9／13

並列化可能性判定のための配列データ依存解析問題のモデル化とシンプレックス法を基とする解法の提案

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，多くの場合 Omega テストより，高速に厳密な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あわせ，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総合的アルゴリズムである．

Modeling of Array Data Dependence Analysis Problem for Parallelization and Proposal of Its Solving Algorithm 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. Several dependence analysis tests on array data have already been proposed. Each test cannot avoid the trade-off between its speed and exactness. Among conventional tests, Omega test is well known as an exact test. 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 linear programming and exhaustive solution search method. This algorithm also includes the features of GCD test, Banerjee test, and Separability test.

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．その理由として，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上げられる．しかし，ループ内の変数に依存関係がある場合，ループボディ毎に並列処理させるとループ繰り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．よって，並列化を行う時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．

†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

配列データの依存解析手法は既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．

文献 4)，5) では，多数の実プログラムの依存解析問題について統計的に調査している．その結論として，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[6] に対して約 63 倍，文献 5) では Perfect Club Benchmarks(以下，PB)[7] に対して約 22 倍，LAPACK[8] に対して約 13 倍，時間がかかることが明らかにされてされている．

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一つであるシンプレックス法[9] と整数解の全探索

```
do i_1 = lb_1, ub_1
  do i_2 = lb_2, ub_2
    ···
    do i_n = lb_n, ub_n
      A(f_1(i_1,···,i_n),···, f_k(i_1,···,i_n)) = ···
      ··· A(g_1(i_1,···,i_n),···, g_k(i_1,···,i_n))···
    end do
    ···
  end do
end do
```

図 1 k 次元配列 A に対する 2 参照を含む一般的な完全入れ子ループ
Fig. 1 A general perfectly nested loop fragment with a pair of references of k-dimensional array A.

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 Laputa テストの流れを説明する．4 章で Laputa テストと Omega テストの解析速度と精度を測定する．5 章では 4 章の測定結果に関して考察する．最後に第 6 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 2 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，繰り返しを考慮する必要がある．異なった繰り返し間でのデータ依存を**ループ繰り越し依存**と呼ぶ．これに対し，同一の繰り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 配列データ依存解析問題の定式化

2.1 節で述べたループ繰り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一組の配列の同じ要素を参照するか否かを調べる必要がある．つまり，対象の配列が属するループが繰り返す間，すなわち，それぞれの制御変数が下限値から上限値まで変化する間に添え字式の値が等しくなることがあるか否かを解析すればよい．異なった繰り返し間での依存関係も解析する必要があるため，2 つの添え字に含まれるループ制御変数 “i” は，独立した変数として扱う必要がある．よって，本論文では，“i_1” と “i_2” と表記する．

図 1 の配列データ依存問題は，2n 個の変数 (i_1_1, i_1_2, $\cdots$,i_n_1, i_n_2) からなる k 個の方程式 (式 (1)) を満たす整数解が，対応する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 (2)，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，式 (1) の方程式と式 (2) の不等式からなる「系」の全条件を満たす整数解を求解する．

$$\begin{aligned} f_1(i\_1_1,\cdots,i\_n_1) &= g_1(i\_1_2,\cdots,i\_n_2), \\ &\cdots \\ f_k(i\_1_1,\cdots,i\_n_1) &= g_k(i\_1_2,\cdots,i\_n_2), \end{aligned} \tag{1}$$

式 (1) 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．ここで，f_i と $g_i(1 \le i \le k)$ はすべて，整数係数の線形式に限定する．

$$\begin{aligned} lb\_1 \le i\_1_1, i\_1_2 \le ub\_1, \\ \cdots \\ lb\_n \le i\_n_1, i\_n_2 \le ub\_n, \end{aligned} \tag{2}$$

また，式 (2) の上下限式については，ここでは次の形に限定する．まず，最外ループの上下限式 lb_1 と ub_1 は整定数，lb_j と $ub\_j(2 \le j \le n)$ はいずれも図 1 の多重ループで，より外側のループ制御変数 $i\_m(1 \le m \le j-1)$ に関する整数係数の線形式とする．したがって，最外側以外のループ制御変数についても，式 (2) を使って，外側のものから順に，取りうる範囲を整数で押さえ込むことができる．

線形性の原則によって，i_1_1，i_1_2 をそれぞれ $x1$，$x2$ に，i_2_1，i_2_2 を $x3$，$x4$ などに変数名を変更し，式 (1) の方程式を式 (3) に標準化することが可能である．同様に式 (2) の不等式も標準化することが可能である．

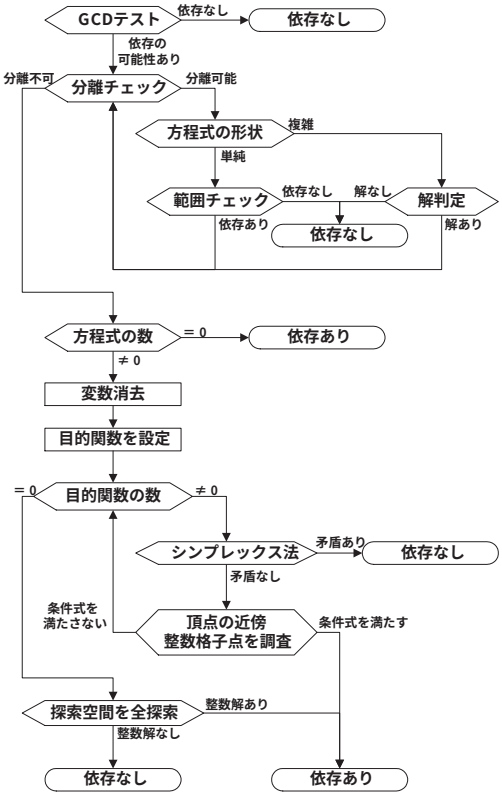
$$\sum_j a_{ij} * x_j + b_i = 0 \tag{3}$$

$(a_{ij}, b_i$: 整定数 $1 \le j \le 2n; 1 \le i \le k)$

3. Laputa テスト

図 3 は Laputa テストの解析の流れの概略図である．Laputa テストでは，まず，入力された方程式に対し，GCD テストを行う．次に，方程式と上下限式に含まれる変数を調べ，問題の一部を簡単な問題として分離可能か否かをチェックする．分離できた部分問題に対しては，方程式の形状により範囲チェックと解判定を行う．この処理は Banerjee テストとほぼ同等の機能となる．ここまでの処理で，比較的簡単な問題に対しては，依存の有無が判定できる．

分離できなかった複雑な問題に対しては，まず，問題に含まれる変数の数を減らすために，方程式を用いて，ループ上下限式から変数消去を行い，シンプレックス法と全探索を行う．シンプレックス法は，線形計画問題をガウスの消去法の基本操作である枢軸 (ピボット) 選択と前進消去を用いて解く方法である．シンプレックス法を適用することにより，(実数) 解空間の頂点座標を求めることができる．次に各頂点座標近傍の整数格子点が整数



captionLaputa テストの流れ
Fig. 1 Flow of Laputa Test

表 1 実験環境
Table 1 Experimental environment

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

解を持つか否かを調べる．整数解とならなかった場合は，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．

4. 評　　価

本章では，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[6] と Perfect Club Benchmarks (以下，PB)[7] 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．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[10] を用いた．

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，全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，

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

	Laputa[ms]	Omega[ms]
(1)	0.006	2.293
(2)	0.038	2.903
(3)	0.042	13.549
(4)	0.186	3.773
(5)	0.231	20.346
(6) (5000 回以下)	0.553	16.549
(6) (5000 回以上)	387.17	32.339
(7) (1000 回以下)	1.056	2.396
(7) (1000 回以上)	242.25	18.320

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	8156	-	-
	(1)〜(4)	7594	-	-
	(5)	550	0.182	7.189
	(6)	0	-	-
	(7)	12	0.182	1.896
Perfect Benchmark	総数	19459	-	-
	(1)〜(4)	19330	-	-
	(5)	129	0.277	16.797
	(6)	0	-	-
	(7)	0	-	-

実際にはまったく意味のないプログラムも含まれる．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) 分離した問題全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

大分類ごとに解析時間の平均を求め，**表 2** にまとめた．全探索を行う大分類 (6)，(7) に含まれる問題に対しては探索回数を求め，(6) に対しては 5000 回以上と以下に，(7) に対しては 1000 回以上と以下に分けて平均解析時間を求めた．また，Laputa テストと Omega テストの依存解析結果は全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．

Linpack と PB に対する調査は，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 Laputa テストの核であるシンプレックス法と全探索部分と Omega テストを比較するため，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．

5. 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 テストと同等の機能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わかる．

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 88 倍高速に解析が可能であることがわかる．

表 2 の大分類 (6) の結果から，全探索の結果，依存がある場合は 5000 回以上の探索回数では Omega テストよりも遅くなることがわかった．しかし，探索回数が 1000 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，解空間の極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，実際には存在しないと考えられる．

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わかる．Laputa テストは 1000 回以上の探索回数で Omega テストより遅くなることがわかった．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．しかし，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表 3 に示したとおり，少ない．以上から，大分類 (6)，(7) 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．

次に表 3 から,「依存あり」となった問題で，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．従って，一般にも「依存あり」となる問題は，この大分類 (5) である確率が極めて高く，解空間の頂点近傍に整数解が見つかるので，全探索する必要がないことがわかる．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，Omega テストより約 40 倍〜約 60 倍高速である．また，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．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．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わかる．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．

6. 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，アルゴリズムについて説明した．本手法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わかった．しかしながら，極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*, Frontier, ACM Press, New York, New York, 1st edition (1990).
2) Banerjee, U.: *DEPENDENCE ANALYSIS*, Loop Transformations for Restructuring Compilers, KLUWER ACADEMIC, Norwell, Massachusetts, 1st edition (1997).
3) Pugh, W. and Wonnacott, D.: Eliminating False Data Dependences using the Omega Test, *Proceedings of the ACM SIGPLAN 1992 conference on Programming language design and implementation*, San Francisco, California, SIGPLAN, ACM, ACM Press, pp. 140–151 (1992).
4) Psarris, K. and Kyriakopoulos, K.: Data Dependence Testing in Practice, *1999 International Conference on Parallel Architectures and Compilation Techniques*, Newport Beach, California, IEEE, pp. 264–273 (1999).
5) Psarris, K. and Kyriakopoulos, K.: The Impact of Data Dependence Analysis on Comiplation and Program Parallelization, *Proceedings of the 17th annual international conference on Supercomputing 2003*, San Francisco, CA, pp. 205–214 (2003).
6) : http://www.netlib.org/linpack/.
7) Berry, M., e. a.: 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).
8) : http://www.netlib.org/lapack/.
9) 小野勝章: 計算を中心とした線形計画法, 日科技連 (1996).
10) : Frameworks and Algorithms for the Analysis and Transformation of Scientific Programs. http://www.cs.umd.edu/projects/omega/.