プログラミング 6－3
（1996. 3. 26）

スカラマシン向けコンパイラにおける配列データフロー解析の評価

飯塚孝好
iitsuka@sdl.hitachi.co.jp
日立製作所システム開発研究所　第3部
〒215　川崎市麻生区王禅寺1099

配列データフロー解析は、自動ベクトル化コンパイラの基盤技術として発展してきたが、スカラマシン向けコンパイラでは十分な評価が無い。本稿では、スカラコンパイラ向けに開発した配列データフロー解析の評価結果を示す。解析手法としては、線形依存システムの厳密解法であるオメガテストを用い、高速化のために近似解法であるSIVテストを併用した。SPECfp92では、1.9%のコンパイル時間増加で、実行性能が14%向上した。SIVテストはコンパイル時間削減に有効であった。更に、wave5は、オメガテストにより5%高速化された。

EVALUATION OF AN ARRAY DATAFLOW ANALYSIS METHOD IN A SCALAR COMPILER

Takayoshi Iiitsuka
iitsuka@sdl.hitachi.co.jp
3rd Research Dept., Hitachi Systems Development Laboratory
1099 Ohzanji, Asao, Kawasaki, 215 Japan

Array dataflow analysis has been evolved as foundation of automatic vectorizing compilers. But there was not enough evaluation on scalar compilers. This paper presents an evaluation of an array dataflow analysis method developed for an scalar compiler. We uses exact analysis method, Omega test, to get high precision and approximate method, SIV test, to speed up analysis. Evaluation with SPECfp92 indicates that compile time increases only 1.9%, and execution performance increases 14%. SIV Test is effective for speed up of analysis. Using Omega test, wave5 speed up 5%.

1 はじめに

b 配列データフロー解析は、主に自動ベクトル化コンパイラの基盤技術として発展してきた[1]。ベクトル化コンパイラでの主要最適化であるベクトル化では、スカラ実行に較べて少なくとも数倍、最大で数十倍程度の性能が得られる。よって、ベクトル化の基盤技術である配列データフロー解析の効果は非常に大きい。また、速度向上比が大きいために Amdahl の法則の影響が大きく、最適化未適用部分を極限まで減らす必要がある。よって、配列データフロー解析では、最大限の解析精度を得るために、解析のコストを無視することも可能であった。

これに対して、ワークステーションなどのスカラマシンでは、最適化による速度向上比がベクトルマシンほど大きくないため、最適化が十分に効かない部分が多少残っていても全体性能への影響は少ない。また、配列データフロー解析を用いた最適化の効果自体、ベクトル化ほど突出していない。更に、配列データフロー解析の精度が多少低くても、ある程度の最適化が可能である場合が多いので、致命的な性能低下に繋がることが少ない。これらの理由から、スカラコンパイラでは解析時間と効果のバランスを考えて配列データフロー解析を設計する必要があると考えられるが、これまで定量的な評価が不十分であった。

本稿では、データ局所性や命令レベル並列性を向上するための各種ループ構造変換など、配列データフロー解析を用いた最適化を強化したスカラコンパイラをベースに、配列データフロー解析の定量的な評価を示す。

2 ベースとなるスカラコンパイラ

ベースとなるスカラコンパイラは、大きく3つのフェーズから構成される。HOPT ではループ構造変換を主体としたプログラム構造変換 [2] を行い、MOPT ではループ不変式移動・共通部分式削除などの伝統的最適化を行い、LOPT ではレジスタ割り付け・命令スケジューリングなどの命令語レベル最適化を行う。配列データフロー解析は、これら3つのフェーズ全てで利用可能とした。各フェーズでの配列データフロー解析の主なクライアントは次の通りである。

- HOPT：ループ交換、外側ループ展開、ループタイリング (ブロッキング)、ループ分配。
- MOPT：ループ不変式移動、配列要素共通化・変数化。
- LOPT：ソフトウェアパイプライニング、大域・局所スケジューリング。

HOPT の最適化は多重ループに跨って実施され、ループ内に解析不能な依存が1つでもあるとうまく最適化できないため、高精度の解析を必要とする。なお、HOPT では、ループ内の全依存ベクトルの集合が求まれば十分な最適化が多いため、解析の処理量は少なく、最適化が有効なループのみを HOPT が選択してオンデマンドで解析するため、処理回数も少ない。

逆に、MOPT および LOPT では、多重ループに跨った最適化は少なく、(内側ループを含まない) 各ループに対する解析のみで十分である。また、解析精度が多少低くても最適化できる場合が多い。しかし、各参照点間の依存情報を設定する必要があり処理量が大きく、解析対象がサブルーチン全体になるため処理回数も大きい。

3 配列データフロー解析の設計

2章の議論から、配列データフロー解析の設計に当たっては、解析精度向上と解析の高速化を同時に達成する必要があると考えた。

3.1 解析精度向上

複雑なループ構造に対して高精度の配列データフロー解析が出来るよう、以下のように解析精度向上を図った。

(1) オメガテストの実装

オメガテスト (Omega test)[3] は、Maryland 大学の Pugh が考案した依存解析手法であり、整数線形方程式・不等式系である依存システムの厳密解を高速に求めることが出来る。解析精度という意味では、その

名前が示す通り、これ以上のものはあり得ない最終版といえる[1]。解析時間は、最悪の場合は指数関数オーダーだが、実際のプログラムでは平均的に線形オーダーで解析できる [3]。4章に概要を記す。

(2) 前方代入／帰納変数代入の実施
配列の添字内にループ制御変数以外の変数が出現する場合、その変数の値が分からないと依存システムの正確な解が求められない。そこで、このような変数は、前方代入や帰納変数代入を用いて、ループ制御変数やループ不変変数の式に置き換える [1]。

(3) ループ上下限式の一般化
ループ構造変換の一種であるループタイリング、ユニモジュラ変換[2]などでは、最適化後ループのループ制御変数の上下限式に min、max、定数除算などの演算が現れる [7]。そこで、これらの演算が現れても正確な解析できるようにした。

3.2 解析の高速化

解析回数が多くてもコンパイル性能が低下しないよう、以下のように解析の高速化を図った。

(1) SIV テストによるふるい落とし
解析精度向上のために導入したオメガテストは、依存システムの厳密解を求める手法としては高速だが、近似テストに比べるとかなり遅いと予想された。また、プログラムに出現する依存関係は、もっと単純で高速な近似テストで解析できる場合が多い。そこで、オメガテストの適用前に、予め、高速な近似テストである SIV テストを適用することにした。SIV テストで依存があると判定され、しかも十分な解析精度が得られない場合に限って、オメガテストを適用する。SIV テストの概要は 5章で紹介する。

(2) 最適化毎のインタフェース設定
配列データフロー解析の結果は、正確には、各参照点の組の間で生じる依存の種類、依存ベクトルの集合、依存ベクトル集合の方向ベクトル別の分解によって表されるが、各最適化では、使わない情報も多い。
そこで、各最適化毎に別々のインターフェースを設定し、必要な情報のみを解析するようにした。これにより、不要な情報に対する解析時間、解析結果の設定時間、保持メモリが節約され[3]、解析処理が高速化されると考えた。ただし、全ての最適化毎に別々のインタフェースを用意するのは無駄が多いので、6章で示す様に数個のインタフェースを用意した。

4 オメガテスト

オメガテスト (Omega Test)[3] は、Fourier-Motzkin 変数消去法 [5] を拡張して、整数解の存在判定を厳密かつ高速に行う手法である。依存ベクトルの方向ベクトル別の分解が必要な際に、方向ベクトル制約を付加せずに依存システムを単純化し、この単純化した結果を元に方向ベクトル別の分解を行うことにより、解析の高速化を図っている。更に、線形不等式のハッシングなど、高速化を意図した実装上の工夫を多数盛り込んでいる。

なお、オメガテストのソースプログラムは公開されているが、SIV テストとの比較に際してプログラミング上の細かなテクニックによる差を除くため、新たな実装を行った。ただし、論文に書かれた実装上の工夫は全て実装してある。

5 SIV テスト

実際のプログラムではループ内の配列添字は簡単な場合が多く、殆どの場合 (97%)、ループ制御変数が 0 個 (ZIV:Zero Induction Variable) か 1 個 (SIV:Single Induction Variable) の場合に帰着できる [6]。複数のループ制御変数を同時に考えなければならないのはわずか 3% である。そ

[1] ただし、ループ制御変数の係数が定数でないなど、依存システムが整数線形等式・不等式系で表現できない場合は、オメガテストでは扱えない。

[2] 設計段階では実装予定だったが、現在未実装。

[3] 参照点の組毎に設定する情報はメモリ量が大きく、また、解析結果の設定自体にも時間がかかる。

こで、単純な添字に対する解析を高速化するために、SIV テストを併用した。

実装した SIV テストは、文献 [6] の解析手法を単純化したものであり、配列添字の各次元毎に別々に依存解析を行う。ただし、文献 [6] と異なり、方向ベクトルと距離ベクトルを統合した依存ベクトル [7] を用い、各次元での依存解析結果を元に、依存ベクトルの範囲を縮小していく。縮小の結果、何れかの次元が ϕ になった場合に依存が無いと判定する。なお、実装した SIV テストでは、ループ上下限に min, max, 整数除算が含まれる場合も解析できる。

添字の各次元の依存解析では、基本的に、ZIV テスト (2 つの添字が定数の場合) と Strong SIV テスト (ループ制御変数の係数が同じ場合) と Weak-zero SIV テスト (一方の添字が定数の場合) のみを実装した。処理が複雑な MIV テストや Delta テストは実装せず、これらの処理はオメガテストにまかせることにした。ただし、比較的頻度の高い三角行列アクセスに対しては、下記の例で示す簡単なヒューリスティックを組み込んだ。

下記のような三角ループ内での $A(J)$ と $A(I)$ の間の依存判定を考える。

```
DO I = 1, N
  DO J = I+1, N
    A(J) = ... (S1)
    A(I) = ... (S2)
  END DO
END DO
```

上記で、S1 から S2 への依存の依存システムを作成すると、次の関係が導かれる：

```
I  < I +1 <= J
I' < I'+1 <= J'
J = I'
```

上記より、次の関係が導ける。

```
I < I'
J < J'
```

よって、依存ベクトルは (+, +) となる。上記の考察を多少一般化し、内側ループのループ制御変数の下限式 (あるいは上限式) が「外側ループのループ制御変数±定数」の場合には、依存システムを作成せずに、直接依存ベクトルを縮小することにした。

SIV テストで依存が無いことを判定できない場合、オメガテストを適用する。しかし、単純添字が多いためオメガテストを実施しても精度が向上しない場合が多い。そこで、次の何れかを満たす場合はオメガテストを抑止している。

(1) 全ての次元で添字解析が正確である。
全ての次元で添字式の組が ZIV テストか Strong SIV テストでに解析され、依存距離が正確に求められた場合。以下、確定依存と呼ぶ。

(2) 依存ベクトルの各次元での方向が 1 つに定まっている。
SIV テストで依存ベクトルが十分に絞り込まれている事が分かる。オメガテストの適用により依存が無いことを判定できる場合もあるが、可能性が少ないと考えた。

6 インタフェースのチューニング

最適化とのインタフェースを幾つかに分類し、各最適化に不要な解析処理を削減することにした。以下では、その代表的なものを示す。

(1) ループネスト内の全依存ベクトル集合。
多くのループネスト最適化では、ループ内の全ての依存に対する依存ベクトル集合の和集合があれば十分であり、依存の種類や、依存を生じる参照点間での依存エッジは不要である。これにより、参照点間での依存エッジ設定に掛かる処理時間・メモリが削減される。

(2) 参照点間のオンデマンド解析。
例えば、ループ分配の適用可能条件は、分配前のループにおいて「分割後に後続するループに属する文」から「分割後に先行するループに属する文」へのループ運搬依存が無いことである。予め各文内の各参照点間に依存エッジを張っておけば、このような判定は容易だが、無駄が多い。そこで、与えられた 2 つの参照点に対してオンデマンド解析を行うインタフェースを用意した。各オンデマンド解析では、特定の最適化に必要な情報のみを解析する。

(3) 中間語の参照点間での依存エッジ
ループ不変式移動などの古典的最適化では、基本的に、フロー依存に対する依存エッジが張られていれば十分であり、依存ベクトルは不要である。なお、運搬ループに対する依存距離(の下限値)が分かると、参照点間に配列の定義があっても共通式削除が出来る場合があので、依存エッジには、運搬ループと依存距離の下限値を付加することにした。

7 評価

本章では配列データフロー解析の効果について、実行性能及びコンパイル性能の面から評価する。評価対象としては、SPECfp92 のうち 12 本の Fortran プログラムを用いた。

表 1に、配列データフロー解析が有る場合の性能向上率、解析時間(%)、依存解析の回数・検出依存数、1 組の参照点間の 1 回の解析当たりの処理時間(μ秒 / 回)を示した。

本表で、AFDA は配列データフロー解析を、SIVT は SIV テストを、OT はオメガテストを、「確定」は SIV テストで検出された依存のうち確定依存の数を表す。

7.1 実行性能

12 本のプログラムのうち、mdljdp2, wave5, tomcatv, mdljsp2, swm256, su2cor, hydro2d, nasa7 の 8 本で実行性能が向上している。最大で 56%(su2cor)、12 本で平均すると 14% の性能向上である。14% という数字は、スカラコンパイラの性能向上率としては大きく、スカラコンパイラでも配列データフロー解析が重要である事が分かる。

なお、SIV テストで十分な精度が得られたと判断した場合、オメガテストを抑止しているが、比較のために、オメガテストを強制的に適用して、性能を測定した。その結果、強制的な適用のみでは、性能は変わらなかった。また、オメガテストを抑止した場合の性能も測定したが、有為な差は見られなかった。

また、MOPT および LOPT では、多重ループに跨った最適化は少ないと考え、(内側ループを含まない)各ループに対する解析のみを行っているが、比較のために、多重ループに跨った解析を実施して性能を測定したところ、wave5 の実行性能が 5% 向上した。ただし、この場合、配列データフロー解析の処理時間は 2 倍にかかった(それでも我々のコンパイラでは、コンパイル時間の 4% に過ぎない)。

7.2 コンパイル性能

12 本のプログラムの全てで、配列データフロー解析の処理時間(AFDA)は比較的少なく安定していることが分かる。即ち、最大でも 3.5% であり、平均 1.9% である。

表 2に AFDA の処理時間の内訳を示す。処理項目の内、実際に配列参照点間の添字を調べて依存ベクトルを求める処理は、「SIV テスト」と「オメガテスト」の 2 項目であり、合計しても 12% に過ぎない。よって、解析時間の殆どは、解析のカーネル部分と無関係な処理であった。

表 2: 配列データフロー解析処理時間内訳

処理項目	%
添字・依存情報の複写・削除	28%
変数レベルでの解析	18%
ループ内参照点間解析主処理	13%
アクセス関数呼び出し準備	11%
アクセス関数オーバーヘッド	10%
オメガテスト	7%
SIV テスト	5%
添字・ループ情報作成	4%
依存エッジ設定	2%

7.3 SIV テストとオメガテストの比較

次に、SIV テストとオメガテストの比較を行う。表 1および 7.1節の議論より、次のようなことが分かる。

(1) オメガテスト特有の効果は少ない。
SIV テストの約 1/4(= 33096/135384) で配列データ依存が存在し、検出された依存の約 1/2(= 17627/33096) は確定依存であり、改善の余地が無い。この結果、実際に

表 1: 配列データフロー解析の効果

	性能	解析時間 (%)			依存解析回数・検出依存数					μ秒 / 回	
	向上率	ADFA	SIVT	OT	SIVT		確定	OT		SIVT	OT
spice2g6	0.0%	1.31%	0.00%	0.02%	4320	1070	636	379	317	2	178
doduc	0.0%	1.50%	0.11%	0.00%	21165	1184	1149	6	6	7	152
mdljdp2	18.8%	2.92%	0.03%	0.04%	1332	822	723	81	81	8	170
wave5	28.6%	3.12%	0.15%	0.51%	46126	17512	7009	6460	6077	8	198
tomcatv	5.7%	3.48%	0.24%	0.31%	2606	702	565	114	67	8	228
ora	0.3%	1.74%	0.00%	0.00%	231	6	6	0	0	0	
mdljsp2	15.2%	2.48%	0.07%	0.05%	2643	1135	1063	63	63	8	194
swm256	20.0%	3.52%	0.50%	0.00%	4690	1381	500	0	0	17	
su2cor	55.5%	1.74%	0.07%	0.24%	12102	2967	1875	999	963	4	172
hydro2d	5.3%	1.58%	0.16%	0.00%	13976	1684	1161	0	0	7	
nasa7	38.9%	3.24%	0.52%	0.14%	24478	4282	2789	356	354	11	201
fpppp	0.0%	0.81%	0.01%	0.03%	1715	351	151	86	86	6	285
全体	14.0%	1.93%	0.09%	0.14%	135384	33096	17627	8544	8014	8	195

オメガテストが適用されるのは全体の 6.3% = (8544/135384) に過ぎない。また、オメガテストを適用しても殆どの依存は無くならない (94% = (8014/8544))。

(2) オメガテストは遅く、SIV テストの併用が効果的。
オメガテストの 1 回あたりの処理時間は約 195μ秒 と安定しているが、処理時間は高速近似テストに比べ約 24倍 = (195/8) 遅い。しかも、殆どの場合は高速近似テストで十分な解析精度が得られる。

8 おわりに

本稿では、スカラコンパイラ向けに開発した配列データフロー解析の評価結果を示した。解析手法としては、線形依存システムの厳密解法であるオメガテストを用い、高速化のために近似解法である SIV テストを併用した。SPECfp92 での評価では、1.9% のコンパイル時間増加で、実行性能が 14% 向上することが分かった。また、SIV テストはコンパイル時間削減に有効であった。

更に、我々のコンパイラでは、残念ながら、オメガテストの効果は余り大きくなかったが、少なくとも 1 本のプログラム (wave5) は、5% 高速化された。ただし、オメガテストが有効となるためには、多重ループに跨った解析が必要であり、配列データフロー解析の処理時間が 2 倍に増加した (それでも我々のコンパイラでは、コンパイル時間の 4% に過ぎない)。

参考文献

[1] H. Zima and B. Chapman, *Supercompilers for Parallel and Vector Computers*, ACM Press, 1990.

[2] 本川, 久島, "RISC コンパイラにおけるループ構造変換," 情報処理学会第 5 1 回全国大会論文集, 第 5 巻、pp.45–46, September 1995.

[3] W. Pugh, "The Omega Test: a Fast and Practical Integer Programming Algorithm for Dependence Analysis," *Supercomputing '91*, pp.4–13, 1991.

[4] M. E. Wolf, "Improving Locality and Parallelism in Nested Loops," *Technical Report No. CSL-TR-92-538*, 1992.

[5] G. B. Dantzig, B. C. Eaves, *Fourier-Motzkin elimination and its dual*, Journal of Combinatorial Theory(A), Vol. 14, pp.288–297, 1973.

[6] G. Goff and K. Kennedy and Chau-Wen Tseng, "Practical Dependence Testing," *PLDI '91*, pp.15–29, 1991.

[7] M. E. Wolf, "Improving Locality and Parallelism in Nested Loops," *Technical Report No. CSL-TR-92-538*, 1992.

[8] U. Banerjee, *Loop Transformations for Restructuring Compilers*, Kluwer Academic Publishers, 1993.