命令レベル並列計算機のためのリカレンス演算の高速化手法とコンパイラへの実装

田中 義一† 前島 英雄††

VLIWやスーパスカラプロセッサなどの命令レベル並列計算機において，リカレンス演算部が性能のボトルネックのひとつとなっている．本研究では，リカレンス演算に対する高速化手法として，高階型後退代入法と，ベクトル計算機のために導入した同次型後退代入法の2つを取りあげ，実際のレジスタ割当てを含めたモジュロスケジューラを試作して評価を行った．その結果，後者は前者に比べ，絶対性能および最適な性能を与える展開数付近での性能感度が小さい点で優れていることが分かった．また，現行の低いレベルの並列度を有するスーパスカラプロセッサにおいても有効であることが確認できた．次に，後者のアルゴリズムを最適化コンパイラで自動的に適用するため，パラメータであるループ展開数を求める経験則を見いだし，プロトタイプコンパイラに実装した．その結果，リバモアカーネル中のリカレンス演算を持つ3ループに対し，理想的な性能の95%以上を得ることができた．

Loop Optimization Method for Recurrences on a Processor with Instruction Level Parallelism

YOSHIKAZU TANAKA† and HIDEO MAEJIMA††

Recurrences often result in a performance bottleneck on a processor with instruction level parallelism because of a chain of flow dependent operations. We compare the two recurrence height reduction techniques that are higher order back-substitution and equal order back-substitution, considering the architecuture resources such as the number of registers. The results show that the latter technique is superior because of optimum performance and insensitivity to loop unrolling factor. Next, we present a heuristic that determines the loop unrolling factor considering the register pressure to apply this technique automatically by optimizing compiler. We can get the performance more than the 95% of optimum results in the recurrence loops in the livermore kernel.

1. はじめに

リカレンスを有するループ演算部は，ループ間にわたるフロー依存によって結びつけられたオペランドを含む演算を逐次的に実行する必要があるため，VLIWやスーパスカラプロセッサなどの命令レベル並列計算機[1),2)]の性能のボトルネックのひとつとなっている．例として，**図1**(a)のような一階リカレンス部を含むループを考える．図1(b)に仮想的な計算機におけるオブジェクトコードを示す．まず，ループ1回あたりの実行時間を演算資源の観点から考える．仮に1つのメモリ演算ユニットと1つの浮動小数点演算ユニットを持ち完全にパイプライン化された計算機の場合，3個のメモリ演算と2個の浮動小数点演算があるため，3サイクルより小さくはできない．また，MUL命令の出力がADD命令の入力となり，ADD命令の出力が次のループ繰返しのMUL命令の入力となるリカレンスの関係にあることから，浮動小数点演算のレイテンシが3の場合，6 ($= 3 * 2$) サイクルより小さくすることはできない．すなわち，リカレンス関係がこのループに対する性能を決めている．

いわゆる超並列計算機に対するリカレンスの高速化のためのアルゴリズムは文献[3)~5)]に見られる．ここでは，計算の冗長度を増加させてリカレンスを高速化している．たとえば，巡回縮約では演算数は N から $N \log N$ と増加するが，並列度も1から N と増大するため，演算器が N 個ある計算機の場合，計算時間は $\log N$ にしか比例せず大幅な高速化が達成される．しかし，限られた並列度を持つ計算機向けの最適化は

† 株式会社日立製作所中央研究所
Central Research Laboratory, Hitachi, Ltd.
†† 株式会社日立製作所半導体事業部半導体開発センタ
Semiconductor Development Center, Semiconductor & Integrated Circuits Division, Hitachi, Ltd.

されていない．

筆者ら[6),7)]は，かつて，一階リカレンス専用の特殊ベクトル命令を持つベクトル計算機向けの高速化手法を研究した．そこでは，限られた並列度を持つベクトル計算機向けに，ループ展開とリカレンスに関する演算木の高さ低減を結びつけたアルゴリズムを考案し，性能評価と実測によりその有効性を示した．

最近，VLIW やスーパスカラプロセッサなどの命令レベル並列計算機に対するリカレンスの高速化の研究が行われ始めた．文献 9)，10) では，ループの前回の値の式を繰り返して代入（後退代入）して，依存性を削減する後退代入法が示されている．しかし，これらの論文では，現実の計算機で問題となるレジスタ数の影響などが考慮されていない．

本報告の目的の第 1 は，命令レベル並列計算機において，演算の冗長度の増加と依存性の削減をバランスよく考慮した 2 つのアルゴリズム，上記の後退代入法（ここでは，高階型後退代入法と呼ぶ）と，筆者らがベクトル計算機のために開発した方法（ここでは，同次型後退代入法と呼ぶ）に関してレジスタ資源も考慮した性能評価を行うことにある．第 2 は，後退代入法を自動的に適用するため，パラメータであるループ展開数を求める経験則を見いだし，プロトタイプコンパイラに実装し，性能評価を行うことである．

2. 後退代入法の性能評価

2.1 仮定するアーキテクチャ

評価を行う際に仮定するアーキテクチャを以下に述べる．

a. メモリ演算ユニット $= u$ 個
　浮動小数点演算ユニット $= u$ 個
　整数演算ユニット $= u$ 個
b. 上記の演算ユニットは完全に並列動作可能
c. 上記の演算ユニットは完全なパイプライン（ある資源を占有するサイクル数は 1）
d. 上記の演算ユニットを使用する命令は同時発行可能
e. ロード命令レイテンシ $= 2$
　浮動小数点演算レイテンシ $=$ lat
　整数演算レイテンシ $= 1$
f. 浮動小数点レジスタ $= m$ 個
　汎用レジスタ $= m$ 個

既存のスーパスカラプロセッサでは，メモリ演算ユニットと整数演算ユニットは兼用され，メモリ演算か整数演算のどちらかが実行可能という場合も多い．しかし，メモリ演算にアドレス更新型命令（メモリ演算の後で，有効アドレスを示すレジスタの値に他のレジスタ等で示す値を加えて更新）を持つプロセッサが増えてきている．したがって，アドレス計算のための整数演算コードが不要となるので，仮定 a は現状に合わないわけでない．また，最近の計算機では乗算と加算を連続的に一命令で行う積和命令を具備するものがあるため，この点についても考慮する．

```
do i = 1, n
  a(i) = a(i-1) * b(i)
       + c(i)
end do
```

（a）ソースコード

```
LD    t1, b(i)
MUL   t2, t1, t4
LD    t3, c(i)
ADD   t4, t2, t3
STR   t4, a(i)
bc
```

（b）仮想計算機コード

図 1　リカレンス演算部を持つループ例
Fig. 1　An example of recurrence program.

```
a(1) = a(0) * b(1) + c(1)
do i = 2, n
  a(i) = (b(i) * b(i-1)) * a(i-2)
       + c(i-1) * b(i) + c(i)
end do
```

（a）2 階の高階型後退代入後のソースプログラム

```
LD    t50, b(i)
MUL   t5, t101, t50
LD    t100, c(i)
ADD   t4, t5, t100
MUL   t3, t51, t50
MUL   t2, t3, t200
ADD   t1, t2, t4
STR   t1, a(i)
REN   t200, t1, 2      ai -> ai-2
REN   t51, t50, 1      bi -> bi-1
REN   t101, t100, 1    ci -> ci-1
bc
```

（b）仮想計算機コード（ループ内のみ）

```
do i = 1, f - 1
  a(i) = a(i - 1) * b(i) + c(i)          f-1回はオリジナルプログラムの方法で実行
end do

do i = f, n                              残りのループは f 階の後退代入法で実行
  a(i) = (b(i) * b(i-1) * ・・・ * b(i-f+1)) * a(i-f)
       + b(i) * (b(i-1) * (・・・*(b(i-f+2) * c(i-f+1) + c(i-f+2))
       ・・・ ) + c(i-2)) + c(i-1)) + c(i)
end do
```

（c）f 階の高階型後退代入後のソースプログラム

図 2　高階型後退代入法
Fig. 2　Higher order back-substitution.

2.2 高階型後退代入法と同次型後退代入法

高階型後退代入法[9)]とは，**図 2** に示すように，前回のループの繰返しでの定義値を繰り返し（f 回）代入し，ループ繰返し距離を増加させ（依存性を削減），さらに乗加算の結合・交換法則を用いてリカレンスに関係する演算木の高さの低減を行う方法である．もともと，n 階の線形関係であったプログラムは，より高階な fn 階の線形関係のプログラムに帰着される．ここで，リマップ命令 REN は実際に出力されることのない疑似コードで，ループ繰返し間の値の流れを表現するために導入したものである[8)]．すなわち，第 2 オペランドのレジスタ上（ベース変数）の値を，第 3 オペランドで示されるループ繰返し（距離）後に，第 1 オペランドのレジスタ（リネーム変数）として参照することを意味する．

一般に，ソフトウェアパイプライン化されたループの 1 回あたりの実行に必要な最小時間である最小イ

テレーション間隔[11] MII は，すべてのオペレーションを実行するために使用する資源から導かれる最小イテレーション間隔 ResMII と，リカレンス関係にあるパス（リカレンスパス）に沿った演算レイテンシから導かれる最小イテレーション間隔 RecMII から，MII=max(ResMII,RecMII) で与えられる．

ResMII は，資源 r を使用する命令が Cr 個，この資源が n_r 個であるとすると，以下のように与えられる．

$$\mathrm{ResMII} = \max_{r \in R} \left\lceil \frac{Cr}{n_r} \right\rceil$$

ここで，R はすべての資源の集合，$\lceil n \rceil$ は n 以上の最小の整数を表す．

次に，リカレンスの関係にある命令間のエッジ e が演算レイテンシ le で de 回のループ繰返し間にまたがっているとすると，RecMII は以下のように与えられる．

$$\mathrm{RecMII} = \max_{c \in C} \left\lceil \frac{\sum_{e \in Ec} le}{\sum_{e \in Ec} de} \right\rceil$$

ここで，C はすべてのリカレンスパスの集合，Ec は，リカレンスパス c のすべてのエッジの集合である．すなわち，依存関係に基づく最小イテレーション間隔 RecMII を求めるときは，フロー依存関係から得られる閉路 c（リカレンスパス）について，c 上の全エッジの演算レイテンシの和を c 上の全エッジのループ繰返しまたがり数の和で割ったものを求め，その最大値をとることで与えられる．

これらから求められる最小イテレーション間隔 MII は，ループ 1 回実行に必要な最小時間を示しており，レジスタが十分にあり，コンパイラが適切にスケジューリングできれば達成できる性能である．

上記の式を，図 2 のプログラムの後退代入部に適用した場合，ResMII に関しては，メモリ演算数が 3，浮動小数点演算数が $3*f-1$ で，後者がネックとなること，RecMII に関しては，$a(i)$ から $a(i-f)$ への依存関係がループ繰返し f 回後であることに注意すれば，

$$\mathrm{MII} = \max(\lceil (3*f-1)/u \rceil, \lceil 2*\mathrm{lat}/f \rceil) \quad f > 1$$

で求められる．この式は後退代入を繰り返すことにより，第 2 項のリカレンスから決まる RecMII は小さくなるが，第 1 項の資源量から決まる ResMII は展開数に正比例して大きくなる欠点があることが分かる．

同次型後退代入法[6]は，図 **3** に示すように 2 つのステップからなる．まず，ループ本体に関し f 倍の

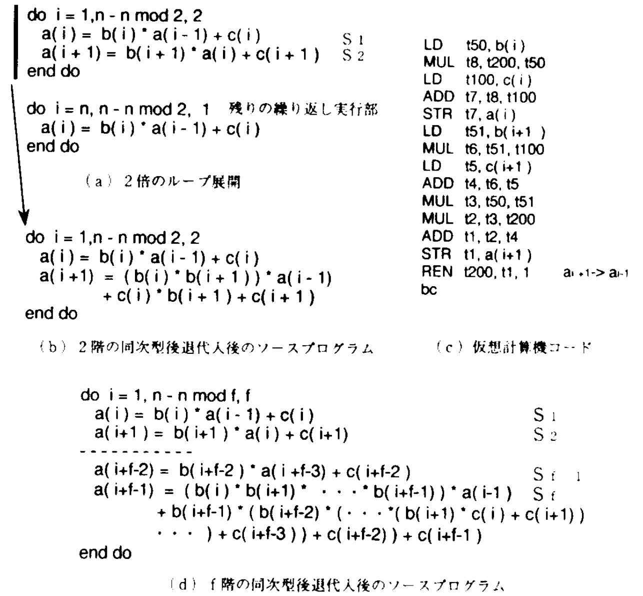


図 3 同次型後退代入法
Fig. 3 Equal order back-substitution.

ループ展開を行う．図 3 (d) の文 S_1 から文 S_{f-1} は，単なるループ展開の文である．文 S_f は，右辺にある $a(i+f-2)$ に対し，文 S_{f-1} から文 S_1 までの右辺の式の代入を順次行い，配列 a に関して $a(i-1)$ のみを含む形に変形し，乗加算の結合・交換則を用いてリカレンスに関係する演算部分を演算木の表現に変換して高さの低減を行って得られた式である．2 階の同次型後退代入法の変換後のソースプログラムを図 3 (b) に示す．もともとのプログラムでは，逐次的に $a(0) \to a(1) \to a(2) \to \cdots$ と求める必要があったが，図 3 (b) のループでは，リカレンスの関係にあるのは文 S_2 だけであり，$a(0) \to a(2) \to a(4) \to \cdots$ と依存関係が半減している．残りの $a(1), a(3), \ldots$ はリカレンスの関係にない文 S_1（文 S_1 は サイクルを構成しない）により求めることになる．このループに対する最小イテレーション間隔 MII は，図 3 (d) に対して高階型後退代入法の場合と同様に考えることにより，

$$\mathrm{MII} = \max(\lceil (5*f-3)/u \rceil, 2*\mathrm{lat})/f \quad f > 1$$

が得られる．なお，全体を f で割っているのは，図 3 (d) のループが f 倍展開されたループであるからである．この式は，展開数が大きくなると，第 1 項である ResMII は一定値に近づき，高階型後退代入法のように展開数に正比例する欠点がないことを示している．

2.3 性能評価

2 つの後退代入法の性能評価を行い，コンパイラによる自動変換への指針を得る．レジスタ数が十分にある場合の性能は，モジュロスケジューリングにおける最小イテレーション間隔 MII の推定において，2.2 節

で示したような評価式を求めることができる．後退代入法が適用される場合，図2(b)や図3(c)に見られるように，プログラム自身の性質としてループ繰返し間にまたがる変数が増加し，必要なレジスタ数が多くなる．このため，レジスタが十分にない場合，変換後のプログラムに対する性能は変換前のプログラムの場合より劣化する可能性がある．そこで，レジスタ数を考慮した評価を行うため，**図4**に示すように，最適化したコード（図2(b)や図3(c)）を入力して，レジスタ割当てとスケジューリングを実際に行ってイテレーション間隔を求めるツールを作成した．

図4で，(1)のループ繰返し間にまたがる変数とは，入力中間語のリマップ命令REN中のオペランドのベース/リネーム変数に示されている．(2)のリカレンスの関係にある文の検出は，(1)で求められた変数を含む文がリカレンスの関係にあるかを判定すればよい．モジュロスケジューリングの(b)においてスケジュール可能なノードは，与えられた資源を用いてできるだけ早く結果を出すため，先行制約グラフにおいてトップからボトムの方向に探す方法をとった．しかし，すでにリカレンスを含むコードがスケジュールされているときは，同じリカレンスパスにあるコードを最優先にスケジュールを行った．ここでは，Cydra 5で採用されている回転レジスタファイル[7]と呼ばれる線形に順序付けされたレジスタ集合と，ループの繰返し制御レジスタによって現在の繰返しに対応したレジスタをさすアーキテクチャを念頭においたため，リマップ命令RENを導入した．しかし，現行のスーパスカラにおいても，必要となるレジスタ数は変わらない．なぜなら，モジュロスケジューリングにおいて，変数の生存区間をII以下にするために，モジュロ展開と呼ばれるループ展開を行うからである．また，通常のスケジュールではレジスタ割当てに失敗した場合はスピルコードを生成するが，ここでは，レジスタが不足した場合にはモジュロスケジューリングの中でIIを増やす方法をとった．また，(5)の最大イテレーション間隔には，通常，ループ繰返しを考慮しないでリストスケジューリングした結果のサイクル数を与える．

（1）ループ繰り返し間にまたがる変数解析
（2）リカレンスの関係にある文集合検出
（3）先行制約グラフの生成
（4）先行制約グラフの各ノードのパスレイテンシの設定
（5）最小イテレーション間隔と最大イテレーション間隔の計算

（6）モジュロスケジューリング
　　最小イテレーション間隔から最大イテレーション間隔までＩＩを
　　変化させながら、以下の処理を最初に成功するまで実施
　（ａ）スケジュール時間ｔｉｍｅの初期化＝０
　（ｂ）先行制約グラフから現在のｔｉｍｅでスケジュール可能なノードｎ
　　　　を選択
　　　　ｉｆ（不在）ｔｉｍｅの更新をして、再度（ｂ）を実行
　（ｃ）ノードｎが資源制約を考慮してスケジュール可能かどうか対応する
　　　　資源予約表のエントリ（ｔｉｍｅ　ｍｏｄ　ＩＩ番目）をみて
　　　　判定し、空いていれば当該エントリの更新
　　　　ｉｆ（使用済み）ｔｉｍｅの更新をして、再度（ｃ）を実行

　すべてのノードがスケジュールされるまで、（ａ）～（ｃ）を実行

　（ｄ）レジスタ割り当ての実行
　　　　失敗した場合、ＩＩを増やして（ａ）からスケジュールを再度実行

　　　　・ループ不変変数　レジスタ１個割り付け
　　　　・ループ可変変数（ループ繰り返しにまたがる同一グループ）
　　　　　　生存区間＜＝ＩＩの場合　生存区間がオーバラップしない
　　　　　　　　　　　　　　　　　　レジスタを探して割り付け
　　　　　　生存区間＞ＩＩの場合　　ceil(生存区間／ＩＩ）個の
　　　　　　　　　　　　　　　　　　レジスタの割り付け

図4　性能評価の続き
Fig. 4　Performance estimation procedure.

以下，1階のリカレンス演算のプログラム（図1）に対する性能評価を行う．**図5**と**図6**に，高階型後退代入法と，同次型後退代入法の性能を，将来の計算機で期待されるようにレジスタが多く（128個）ある場合と，現行のスーパスカラプロセッサなみの32個の場合に関して示す．横軸は各方法の展開数，縦軸はもともとのプログラムに対する性能向上率を示すもので，各折れ線はユニット組数に対応している．なお，仮定している浮動小数点演算レイテンシはlat=3である．また，高階型後退代入法において，レジスタが32の場合，資源組数が4と8の場合は，レジスタ不足のため完全に性能が一致している．

これらの図から，高階型後退代入法に比べ同次型後退代入法の方が，レジスタ数のいかんにかかわらず，次の2点から明らかに優れていることが分かる．すなわち，性能の絶対値が大きいこと，および最適な性能を与える展開数付近での，展開数による性能の感度が小さいことである．コンパイラにおいて自動的に変換を行う場合，コンパイラは最適な展開数を決定する必要がある．最適な展開数を，許されるコンパイル時間で正確に求めることはできないため，後者の性質は特に望ましいものである．

以下では，同次型後退代入法のみに注目して検討を行う．現在のスーパスカラプロセッサでは資源の組が1個（$u = 1$）と小さいが，将来，演算資源を増加させた場合，一般のプログラムにおいても並列度に見合う性能向上のためには，演算資源に応じてレジスタ資源も増加させる必要がある．そこで，演算資源の1組あたりのレジスタ数を16とし，演算資源に比例してレジスタを増加させたアーキテクチャに基づいて検討を行う．**図7**に，この条件のアーキテクチャでの性能を示す．最適な展開を行った際の性能は，資源数1（$u = 1$）の場合，後退代入をしない場合に比べ1.71倍，資源数2のとき2.57倍，そして資源数4のとき4.0倍と大幅な性能向上を達成することができる．資源数1の性能から，現行のスーパスカラプロセッサで

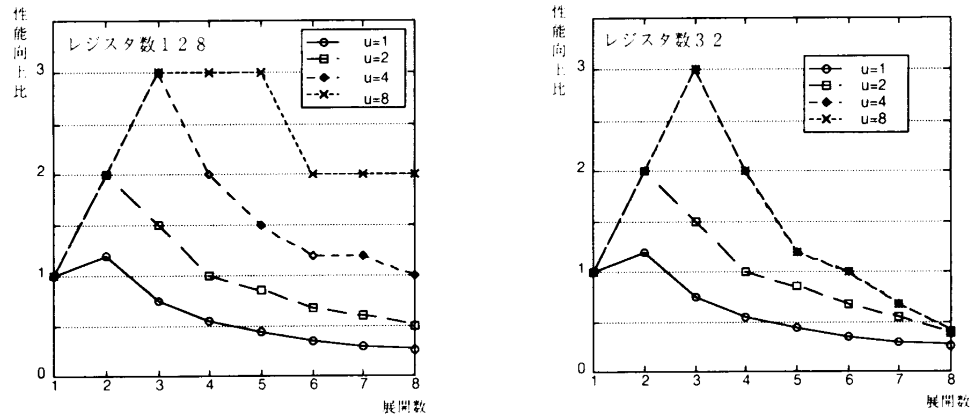


図5 高階型後退代入法による性能向上比
Fig. 5 Performance of linear recurrence with the higher order back-substitution.

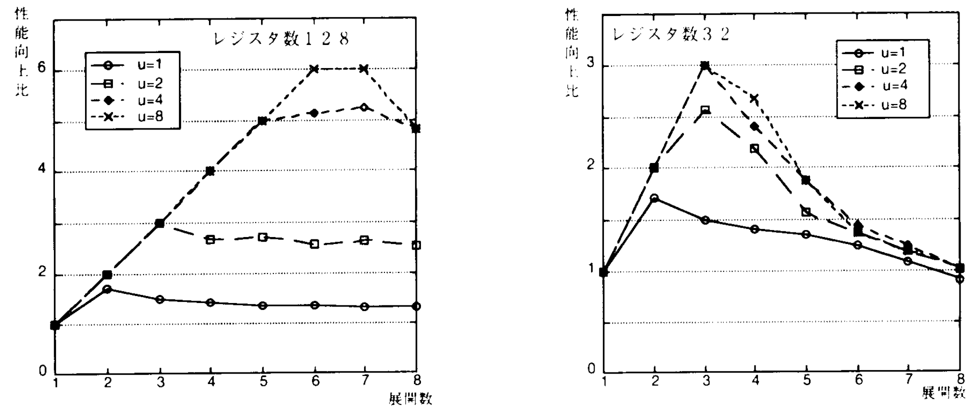


図6 同次型後退代入法による性能向上比
Fig. 6 Performance of linear recurrence with the equal order back-substitution.

も有効な方式であることが確認できる．**図8**は，イテレーション間隔IIを示したものである．もともと，1浮動小数点演算あたり，3 (= 6/2) サイクル要したプログラムが，資源数1のとき1.75サイクル，資源数2のとき1.15サイクル，そして資源数4のときには0.75サイクルに向上する．さらに，図中には，レジスタが無限に存在するとして，2.2節で示した最小イテレーション間隔MIIに基づく性能を示してある．このMIIに基づく曲線と実際の性能を比べることにより，コンパイル時において次のような展開数決定戦略が実務的に有効であることが分かる．

「コンパイル時に，展開数 f の，ResMIIとRecMIIから最小イテレーション間隔MIIを推定し，f を1だけ増やすことにより，推定性能の向上率がたとえば20%未満であるなら，さらなる展開は行わない.」

たとえば，上記の基準で展開数を決定すると，このループにおいて，実際に最大の性能が実現できる．仮に，基準値を20%から10%に下げると，資源数4の場合にのみ，最適値から1回余分に展開を行い，性能は4.0倍の性能から3.75倍の性能に劣化するだけである．

図9は，最近のスーパスカラプロセッサが持つ積和命令を使用した場合の性能を示す．浮動小数点の乗算と加算を連続的に行い，3 * 2サイクルを要したリカレンスパス上の演算が，積和命令で半分の3サイクルとなる．このため，リカレンスによる性能への影響は小さくなることから，後退代入法の効果は小さくなる．特に，資源数が1のときは逆効果となる．しかし，資源数1の場合でも，スーパパイプラインプロセッサのように浮動小数点演算レイテンシが大きいときは，この変換は有効である．**図10**は，他のリカレンス計算の例として，図1のプログラムにおいて，リカレンスオペランドの係数が1（すなわち $B(i)$ を1に置き換えたケース）である部分和計算の性能結果を示す．

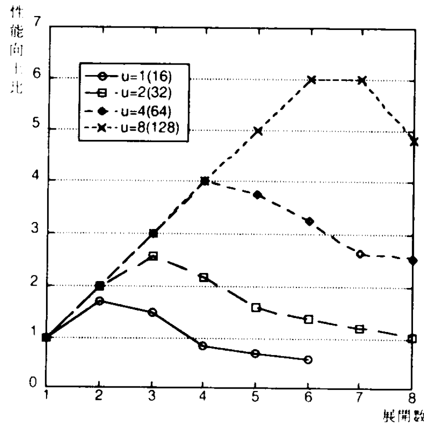


図7 レジスタを比例増加した場合の同次型後退代入法による性能向上比

Fig. 7 Performance of linear recurrence with the equal order back-substitution in case of proportional number of registers.

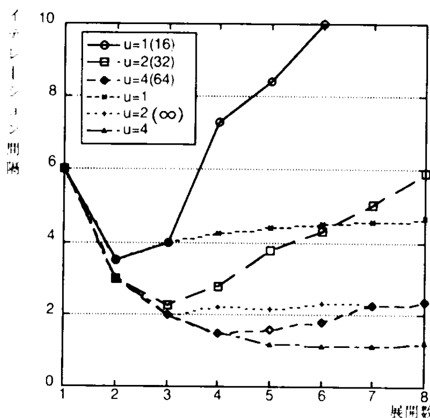


図8 同次型後退代入法におけるイテレーション間隔

Fig. 8 Iteration interval of linear recurrence with the equal order back-substitution.

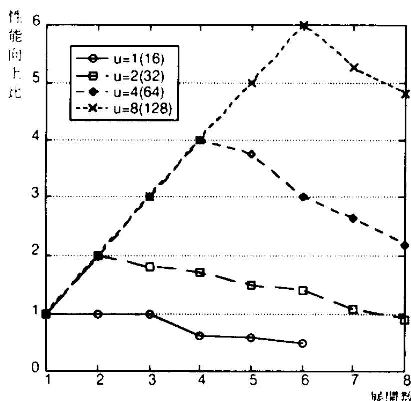


図9 乗加算命令がある場合の性能向上比 (lat=3)

Fig. 9 Performance of linear recurrence using multiply-and-add instruction.

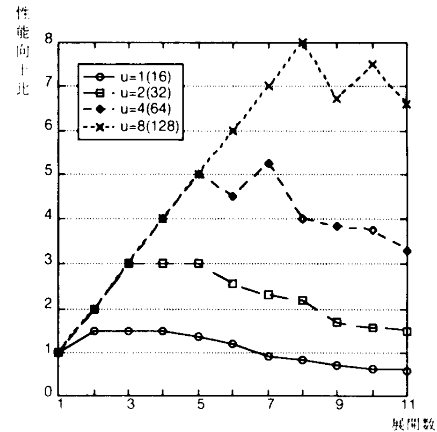


図10 部分和演算の性能向上比 (lat=3)

Fig. 10 Performance of partial sum with the equal order back-substitution.

3. 同次型後退代入法のコンパイラへの実装

3.1 コンパイラ構成

リカレンスを含む演算の同次型後退代入法による自動変換のため，グローバルな伝統的最適化処理の後に，最内側ループのそれぞれに対して以下の処理を順に行うプロトタイプコンパイラを開発した．

1. メモリ演算最適化
2. 演算木の高さ低減
3. 後退代入
4. 演算木の高さ低減
5. メモリ演算最適化
6. 伝統的最適化
7. モジュロスケジューリング

前章では，後退代入とリカレンスパス上のオペランドに関する演算木の高さ低減の 2 つの処理をあわせて後退代入と呼んでいたが，ここでは**図 11** のように両手順を分離する．2 つに分けるのは，インプリメント上，演算木の高さ低減処理は後退代入を適用しないケースにも有効であるためである．以下，図 11 の例を用いて処理の流れを説明する．(a) のループに対し，最適化ののち (b) のような形式で存在すると仮定する．

1 のメモリ演算最適化は，配列要素に関して共通化（同一ループ繰返し，または繰返しにまたがった共通化）を行い，配列要素をレジスタに保持する変換を行う．定数差のループ繰返しで同一となる配列要素を順番に並べ，(c) のようなリマップ命令 REN を導入し，無駄なメモリ演算を削減する．ここで，ベース変数 *btmp* は $a(i)$，誘導されるリネーム変数 *rtmp* は $a(i-1)$ を保持する．この最適化は，後退代入における展開数決定の所用資源量の決定の精度向上に重要で

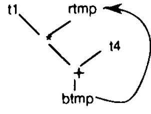
```
do i = 1, n
 a( i ) = b( i ) * a( i - 1 ) + c( i )
end do
```

(a) オリジナルソース

```
do i = 1, n
 t1 = b( i )
 t2 = a( i - 1 )
 t3 = c( i )
 t4 = t1 *  t2 + t3
 a( i ) = t4
end do
```

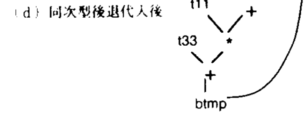
(b) 内部中間語形式

```
do i = 1, n
 t1 = b( i )
 t3 = c( i )
 btmp = t1 *  rtmp + t3
 a( i ) = btmp
 REN  rtmp, btmp, 1
end do
```



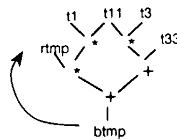
(c) メモリ演算最適化1後

```
do i = 1, n - n mod 2, 2
 t1 = b( i )
 t11 = b( i + 1 )
 t3 = c( i )
 t33 = c( i + 1)
 t5 = t1 * rtmp + t3
 a( i ) = t5
 btmp = t11 * ( t1 * rtmp + t3 ) +  t33
 a( i + 1) = btmp
 REN  rtmp,  btmp, 1
end do
```



(d) 同次型後退代入後

```
do i = 1, n - n mod 2, 2
 t1 = b( i )
 t11 = b( i + 1 )
 t3 = c( i )
 t33 = c( i + 1)
 t5 = t1 * rtmp + t3
 a( i ) = t5
 btmp = ( t11 * t1 ) * rtmp  + ( t11 *  t3  +  t33 )
 a( i + 1) = btmp
 REN  rtmp,  btmp, 1
end do
```



(e) 演算木の高さ低減2後

```
do i = 1, n - n mod 2, 2
 btmp1 = b( i + 1 )
 btmp2 = c( i + 1)
 t5 = rtmp1 * rtmp + rtmp2
 a( i ) = t5
 btmp = ( btmp1 * rtmp1 ) * rtmp  + ( tbtmp1 *  rtmp2  +  btmp2 )
 a( i + 1) = btmp
 REN  rtmp,  btmp, 1
 REN  rtmp1,  btmp1, 1
 REN  rtmp2,  btmp2, 1
end do
```

(f) メモリ演算最適化2後

図 11 例題による最適化の手順

Fig. 11 Successive application of optimization steps.

ある．2 の演算木の高さ低減は，もともとのループに対して，3.3 節で述べるようにリカレンスパス上の演算木に関する高さの低減を行う．ベース変数とリネーム変数を含む文からリカレンスを構成する文を検出し，リカレンスの関係を持つリネーム変数を演算の交換・結合則を用いて木の下方に移動することにより，リカレンスパスの演算レイテンシを短くする．しかし，この例は，(c) に示すようにこれ以上高さを低減できない．3 の後退代入は，リカレンスパス上の演算レイテンシの長さが性能のボトルネックになる場合，後退代入の適用の可能性を解析し，性能が向上すると判断された場合には後退代入変換を行う．(d) に後退代入後のコードが示してある．4 の演算木の高さ低減は，後退代入変換が行われたループに関して，2 と同様の最適化を再度行う．この例では，太字で示すリカレンスを構成する文に関して，(e) で示すように演算木の高さ低減を行い，リカレンスパスに関する演算レイテンシの大きさを半減している．5 のメモリ最適化は，後退代入変換が適用されたループに関して，1 と同様の最適化を再度行い，(f) のような最適化コードを得る．6 の伝統的最適化は，主に後退代入によって新たに生成したコード，特にアドレス計算コードに対するストレングスリダクション等の最適化を行うために存在する．この処理がないと，メモリ演算の最適化を実施しない場合と同様に，展開後の資源所要量が後退代入法で推定した値より大きくなり所望の性能が得られない．

3.2 同次型後退代入法のインプリメント方式

同次型後退代入法をコンパイラで自動的に実現するためには，a. 後退代入が原理的に可能かの判定，b. 最適な展開数の決定，c. コード変換，の処理をインプリメントする必要がある．a と c に関しては文献6) を参照されたい．b の最適な展開数を決定するためには適切な経験則が必要である．演算資源とレジスタ資源がバランスよく実現された計算機を対象とすると，2.3 節の性能評価で述べたような基準を用いればよい．

3.3 演算木の高さ低減のインプリメント方式

演算木の高さ低減は，以下のように行う．ループ内のリカレンスパスを求め，各々のリカレンスパスの演算レイテンシを求め，レイテンシの大きい順に以下に示す 4 つの演算木の組替えプリミティブを，変化がなくなるか，資源から決まる ResMII がリカレンスから決まる RecMII を超えるまで繰り返して行う．

(a) 因子化 1

$$rtmp * t1 + rtmp * t2$$
$$\to rtmp * (t1 + t2)$$

(b) 因子化 2

$$(rtmp * t1 + A) + rtmp * t2$$
$$\to rtmp * (t1 + t2) + A$$

(c) 交換化

$$(A : rtmp + B) + C$$
$$\to (C + B) + A : rtmp$$
$$(A : rtmp * B) * C$$
$$\to (C * B) * A : rtmp$$

(d) 分配化

$$(A : rtmp + B) * C$$
$$\to A : rtmp * C + B * C$$
$$(A : rtmp * B + C) * D$$
$$\to (A : rtmp * B) * D + C * D$$

ここで，$rtmp$ とはリネーム変数，$t1/t2$ は通常のテンポラリ変数，$A/B/C/D$ は一般の演算木で，$A : rtmp$ は演算木 A の中に $rtmp$ を含むことを意味する．(c) の交換化においては，置き換えの対象となる木の中に，リカレンスパスに関するレイテンシの長い他のリネーム変数がないことが重要である．(d) 分配化は，演算量を増加させるので，後のステップで必ず (c) の交換化によりリカレンスパスの演算レイテンシが低減されるときのみに適用を限定する．

表1 ループ1回実行するためのサイクル数
Table 1 Performance of livermore kernels.

Kernel #	浮動小数点演算数	オリジナル $u=1,2,4$	コンパイラによる自動最適化 $u=1$ 16レジスタ	$u=2$ 32レジスタ	$u=4$ 64レジスタ
5	2	6 (0.33 flop/cycle)	4 (0.5 flop/cycle) (1.5倍)	2.3 (0.87 flop/cycle) (2.5倍)	1.6* (1.25 flop/cycle) (3.8倍)
11	1	3 (0.33 flop/cycle)	2 (0.5 flop/cycle) (1.5倍)	1 (1.0 flop/cycle) (3倍)	0.6** (1.67 flop/cycle) (5倍)
23	12	18 (0.61 flop/cycle)	12 (0.92 flop/cycle) (1.5倍)	6 (1.83 flop/cycle) (3倍)	6 (1.83 flop/cycle) (3倍)

真の最適値： *1.5, **0.57

3.4 リバモアカーネルにおける性能例

ベンチマークプログラムとして知られるリバモア24カーネルには，5つのリカレンス関係を持つループがある．そのうち，依存距離がループ繰返しごとに変わらないループは，#5，#11，#23の3つである．これらのループに対して，プロトタイプコンパイラで実現したオブジェクトによるループ1回あたりの実行サイクルの値を**表1**に示す．表において，オリジナル欄は，ソースプログラムに対してFORTRAN文法の指示するとおりの演算順序で実行を行った場合の，また，コンパイラによる自動最適化欄は，本章で述べた方式をプロトタイプコンパイラにインプリメントして最適化した場合の結果を意味する．また，1サイクルあたりの浮動小数点演算数や，本最適化を行わない場合との性能比も示してある．なお，ループ23は，データ依存関係が変化するので，原理的に後退代入を行うことはできない．そのため，資源数を2から増やしても，リカレンスパスが性能のボトルネックとなっていることから，性能の向上はできない．

4. おわりに

本稿では，命令レベル並列計算機におけるリカレンス演算に対する高速化手法として，高階型後退代入法と，ベクトル計算機のために報告者らが提案した同次型後退代入法の2つを取りあげ，実際のレジスタ割当てを含めたスケジューラを試作して評価を行った．その結果，後者は前者に比べ，絶対性能および最適な性能を与える展開数付近での性能感度が小さい点で優れていることが判明した．また，性能解析の際に，最適なループ展開数の決定に関して見い出した経験則をプロトタイプコンパイラに実装し，リバモアカーネル中のリカレンス演算に適用してその有効性を確認した．

今後の課題としては，データ依存関係から後退代入の適用ができない場合のリカレンス演算に関する高速化があげられる．

参考文献

1) Fisher, J.A. and Rau, B.R.: Instruction-level Parallel Processing, *Science*, Vol.253, pp.1233–1241 (1991).
2) Johnson, M.: *Superscalar Microprocessor Design*, Prentice Hall (1991).
3) Stone, H.: An Efficient Parallel Algorithm for the Solution of Tridiagonal Linear System of Equations, *JACM*, Vol.20, No.1, pp.27–38 (1973).
4) Heller, D.: Some Aspects of the Cyclic Reduction Algorithm for Block Tridiagonal Linear Systems, *SIAM J. Numerical Analysis*, Vol.13, No.4, pp.484–496 (1976).
5) Rodrigue, C. (Ed.): *Parallel Computations*, Academic Press, New York (1982).
6) Tanaka, Y., Iwasawa, K., Gotoo, S. and Umetani, Y.: Compiling Techniques for First-Order Linear Recurrences on a Vector Computer, *Proc. the Supercomputing Conference (Orlando, FL)*, pp.74–181 (1988).
7) Gotou, S., Tanaka, Y., Iwasawa, K., Kanada, Y. and Aoyama, A.: Advanced Vectorization Techniques for Supercomputers, *J. of Information Processing*, Vol.11, No.1, pp.22–31 (1987).
8) Rau, B.R., Schlansker, M.S. and Tirumalai, P.P.: Code Generation Schemas for Modulo Scheduled DO-Loops and WHILE-Loops, *Proc. the 25th Annual International Symposium on Microarchitecture (Portland, OR)*, pp.158–169 (1992).
9) Dehnert, J.C. and Towle, R.A.: Compiling for the Cydra5, *The Journal of Supercomputing*, Vol.7, pp.181–228 (1993).
10) Schlansker, M.S. and Kathail, V.: Accelera-

tion of Algebraic Recurrences on Processors with Instruction Level Parallelism, *Proc. the Sixth Workshop on Languages and Compilers for Parallel Computing (Portland, OR)* (1993).
11) Ramarkrishnan, S.: Software Pipelining in PA-RISC Compilers, *Hewlett-Packard Journal*, pp.39-45 (1992).

(平成7年6月8日受付)
(平成8年6月6日採録)

田中　義一（正会員）

1977年東京大学工学部航空学科卒業．1979年同大学院修士課程修了．同年（株）日立製作所入社．以来，同社中央研究所において，スーパコンピュータ向けコンパイラおよびアプリケーションプログラムの研究開発に従事したのち，現在，マイクロプロセッサ用のコンパイラおよびアーキテクチャの研究に従事．現在，同所主任研究員．共著「スーパコンピュータ」（オーム社）．ACM会員．

前島　英雄（正会員）

1971年東京工業大学理工学部制御工学科卒業．1973年同大学院修士課程修了．同年（株）日立製作所入社．同社日立研究所において，制御用計算機の開発に従事した後，マイクロプロセッサ，専用プロセッサなどのマイクロアーキテクチャおよびLSI技術の研究に従事．現在，同社半導体事業部半導体開発センタ主管研究員．IEEE，電子情報通信学会各会員．工学博士．