計算機アーキテクチャ 89－19
（１９９１. 7. １９）

並列処理システム-晴-における条件分岐の先行評価制御方式

山名早人　石崎一明　安江俊明　村岡洋一
yama@muraoka.info.waseda.ac.jp
早稲田大学理工学部

本報告では，条件分岐の先行評価による実行時間の短縮効果について述べ，先行評価の一実行方式として，データ駆動を用いた方式を提案する．条件分岐の先行評価は，VLIW計算機を中心に数多く行われている．しかし，これらの方式では多段に渡る先行評価を行うことができず，十分な速度向上が得られない．これに対して，Boolean Recurrenceを持つ逐次ループでは，先行評価段数を増加させることにより，大きな速度向上が得られる．本報告では，Boolean Recurrenceを持つループに対して，常に条件分岐n段分の先行評価を行う方式をデータ駆動方式により実現する方法を提案する．

A Parallel Execution Scheme of Conditional Branches Using Eager Evaluation for the Parallel Processing System -Harray-

Hayato YAMANA　Kazuaki ISHIZAKI　Toshiaki YASUE　Yoichi MURAOKA

School of Science and Engineering, Waseda University
Room 59-422, 3-4-1 Okubo, Shinjuku-ku, Tokyo 169, Japan

The purpose of this paper is to propose a parallel execution scheme of Conditional branches using eager evaluations. Such a scheme has been proposed on VLIW machines. However, they can not extract enough parallelism from a program because they can not bypass many branches. In this paper, we propose a new execution scheme which enables n-bypassing conditional branches on a dataflow machine. The scheme is applied to boolean recurrence loops to shorten the execution time.

1. まえがき

本報告では，条件分岐を含む逐次型のループの内，Boolean Recurrence[BaGa84]を持つループを並列実行する方式として，データ駆動方式を用いて常にn段の条件分岐を先行評価する手法を提案する．

逐次型ループは，(1)繰り返し間に制御依存関係が無いにも関わらずデータ依存関係によって並列に実行できないループと(2)制御依存とデータ依存関係が相互に絡み合うことによって並列実行できないループの２つに大別できる．

前者のループは，例えばdoacross型ループにおいて各繰り返しの開始までのディレイ[Cytr86]がループ本体の実行時間と等しいループである．この場合は，データ依存関係のみに基づいて実行時間が決定するため，並列化の余地がない．これに対して，後者のループは制御依存とデータ依存を切り放した時に，各々を独立に計算できれば，ベクトル計算機において用いられているマスクベクトルと同様の手法で並列化が可能である．しかし，独立でない場合，すなわち，ある繰り返しで求められたデータが条件分岐によって，次回以降の繰り返しの条件判断に対してデータ依存を持つような場合は，繰り返し間で並列に実行を行うことができない．このようなループをBoolean Recurrence[BaGa84]を持つループと言う．

Boolean Recurrenceを持つループを並列化し，実行時間を短縮するためには，[RiFo72][BaGa84][Uht88]に述べられているように，条件分岐をジャンプして（各条件分岐を先行評価することによって），最終的には必要のない演算までも行わなければならない．[RiFo72]では，いくつかのアプリケーションにを用いてシミュレーションを行い，条件分岐を先行評価することによって，平均51倍（条件分岐を無限に先行評価した場合）の性能向上が得られることを示している．しかし，[RiFo72]では実行方式についての記述はない．これに対して[BaGa84]では，専用ハードウェアによる実現方式を提案しているが，対象となるループは，ループ内に１つの条件分岐を持つ場合に限定され，多数の条件分岐文を持つループに適用できない．さらに，[Uht88]においては，ループ内に１つの条件分岐を持つループを対象とした時の，必要プロセッサ数を算出する方法を示しているが，やはり多数の条件分岐文を内部に持つループには適用できない．

本報告では，Boolean Recurrenceを持つループ一般．すなわち複数の条件分岐を内部に持つループについての条件分岐ｎ段先行実行をデータ駆動計算機を用いて実行する方法を提案する．データ駆動計算機を用いることの利点は，ループ実行時に決定される動的なデータ依存関係をデータフローグラフで表すことによって，各演算を最速のタイミングで実行できる点にある．これをVLIWのようなシングルプロセッサ上で行おうとすると，(1)動的にデータ依存関係が変化することによって生じるデータの定義参照関係を守るために，実行中のあるタイミングで同期をとる必要が生じるため効率が悪い，(2)制御のスイッチングを頻繁に行う必要があり，パイプラインを満たすことができない等の問題がある（３節参照）．

以下では，まず，条件分岐を先行評価することによるプログラムの実行時間短縮の効果について述べる．特に我々がこれまでに提案してきたフローグラフ展開手法[YYKM91]との違いを明かにする．３節では，シングルプロセッサ上での実行上の問題を述べ，４節で，データ駆動計算機を用いたｎ段先行実行方式を我々の提案している－晴－[YMHM88]を例にとって説明する．

2. 条件分岐の先行評価効果

本節では，条件分岐の先行評価によるプログラムの実行時間短縮の効果を調べるために行ったシミュレーション結果について述べると共に，我々がこれまでに提案しているフローグラフ展開による条件分岐の先行実行方式と今回提案するn段先行実行方式の性能比較を行う．

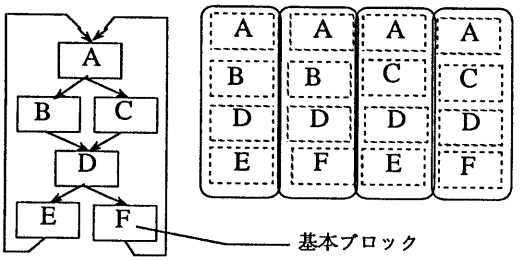
2.1 フローグラフ展開とｎ段先行実行方式

フローグラフ展開[YYKM91]は，図１に示すように，コンパイル時にループ部分を予め条件分岐ｎ段分（プロセッサ数に応じた段数）の展開をした静的なコード（スレッドと呼ぶ）を生成し，ｎ段毎の条件分岐の先行評価を行う手法である．しかし，静的に生成されたコードを用いているために，同一のコードを再利用するためにはｎ段毎の同期が必要となる．これに対して，今回提案する手法は，条件分岐を常にｎ段先行して実行する方式である．これらの動作を比較した図を図２に示す．

2.2 条件分岐の先行評価

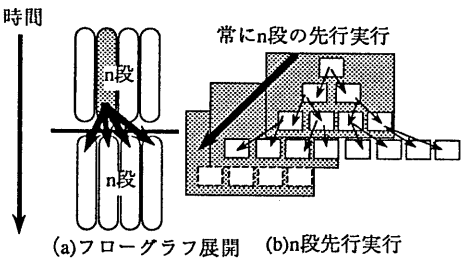
条件分岐を常にｎ段先行評価することによる効果

を調べるために，表１に示す８本の科学技術計算プログラムを用いてシミュレーション評価を行った．また，シミュレーションで用いた仮定を表２に示す．



(a) control flow graph (b) フローグラフ展開後のスレッド

図１ フローグラフ展開



(a)フローグラフ展開 (b)n段先行実行

図２ フローグラフ展開とｎ段先行実行方式

表１ 対象プログラム

	基本ブロック総数	行数	基本ブロックの粒度(行)	使用データ規模
【収束判定プログラム】				
LM(一次関数極小化) DOALLを含まず	133	399	3.0	逐次実行3万ステップまで実行
TS(実超越方程式)	63	179	2.8	
MI(多変数関数極小化) DOALLを含む	259	780	3.0	
NO(関数2乗和の極小化)	139	388	2.8	
【ループ実行プログラム】				
HB(固有ベクトル逆変換)	143	337	2.4	50*50
BL(平衡化)	45	169	2.4	
LT(実3項行列のガウスの消去法)	74	235	3.2	
LX(クラフト法)	119	387	3.3	

表２ シミュレーションにおける仮定

計算機資源 ： 無限大
演算実行時間 ： 全て１単位時間
データ通信時間 ： ０
計算実行モデル ： データ規模５０×５０
ベクトル方式（DOALL型ループのみを並列処理）
データ駆動方式（データが揃った時点で演算を実行）
フローグラフ展開方式（先行評価n段で同期）
n段先行実行方式（常時n段の先行評価）
ただし，いずれもDOALLループを越えて先行評価をしない．

なお，シミュレーションにおける計算機資源無限大の仮定は，ベクトル部分の実行時間を除いた部分の実行時間の推移を求めるためである．すなわち，ベクトル部分の計算を１単位時間で行うことを仮定することによって，（シミュレーションの結果得られた実行時間≒逐次ループ部分の実行時間）と考えることができる．これによって，逐次ループ及びその他の逐次部分の実行時間が先行評価によってどの程度時間短縮されるかを求められる．

まず，図３にデータ規模（扱うデータの大きさ）をパラメータとした処理速度比の推移を，ループ実行プログラムについて示す．先行評価をしない場合を基準とした時の処理速度向上比（先行評価をしない場合を１とし，先行評価段数＝無限大時の処理速度向上比）を示す．**図３**より，データの規模が大きくなるにつれ，(1)ｎ段先行実行の効果が顕著に表れてくるプログラムと，逆に(2)ｎ段先行実行の効果が薄れてくるプログラムが存在することがわかる．これは，データ量が増大すると，プログラム全体に占める逐次ループの実行時間の割合が増加し，(1)逐次ループ内に条件分岐が存在すればｎ段先行実行の効果が表れるが，(2)条件分岐が存在しない場合には，ｎ段先行実行の対象とならず，逐次ループの実行時間が短縮されない．従って，プログラム全体としてみた時の処理速度向上率が下がることに起因している．すなわち，条件分岐の先行実行は，逐次ループが内部に条件分岐を持っており，さらにその逐次ループの実行時間が全体の実行時間に占める割合が大きいほど効果が大きいことがわかる．

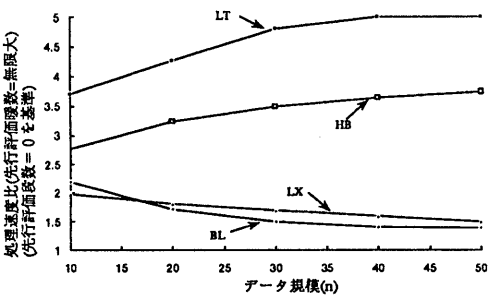


図３ データ規模と処理速度比

次に，ベクトル方式（DOALL型ループのみを並列処理）を基準とした場合のｎ段先行実行方式の処理速度向上比を図４に示す．すなわち，逐次部分の処理速度向上率を示す．ただし，図３においてデータ規

模が増大するにつれ，先行実行の効果が低下する２つのプログラムは割愛した．この図より，５段先行実行を行うことによってデータ駆動方式に比較して約2.2-4.4倍程度の処理速度向上がプログラム中の逐次部分に対して期待できることがわかる．なお，この結果は[RiFo72]における結果（５段先行評価時平均４倍の性能向上）とほぼ等しく，シミュレーションが正しく行われていることがわかる．５段の先行実行を行うためには，最大O（2^6）のプロセッサが必要であるが，実現できないプロセッサ数ではない．また，５０段展開した場合の結果も合わせて示した．

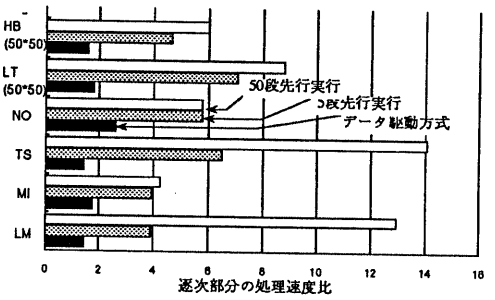


図４　ｎ段実行方式による処理速度向上

最後に，フローグラフ展開とｎ段先行実行の処理速度の比較を行う．図５に比較結果を示す．結果より，ｎ段毎に同期をとるフローグラフ展開とｎ段先行実行の処理速度比は最大３０％程度であることがわかる．しかし，実際にはフローグラフ展開では，ｎ段毎の同期オーバヘッドが発生するのに対して，ｎ段先行実行方式では常にｎ段の先行実行を行うためオーバヘッドが陽に表れない．従って，３０％の差はさらに大きくなるものと考えられ，ｎ段先行実行の有効性が理解できる．また，特に処理速度が飽和に達する５－１０段付近での差が最も大きい．実際のプロセッサ数を考えた場合（最大１０００台程度）の先行評価段数が５－１０段程度であると考えると，ｎ段先行評価がフローグラフ展開に勝ることがわかる．

以上をまとめると，(1)ｎ段先行実行方式は，我々が従来提案しているフローグラフ展開に比較して３０％程度の処理性能向上を得ることができ，(2)今回評価の対象となったプログラムでは，その逐次部分を約2.2-4.4倍（ｎ＝５）高速化できることがわかる．この時必要なプロセッサ数は最大O（2^6）である．

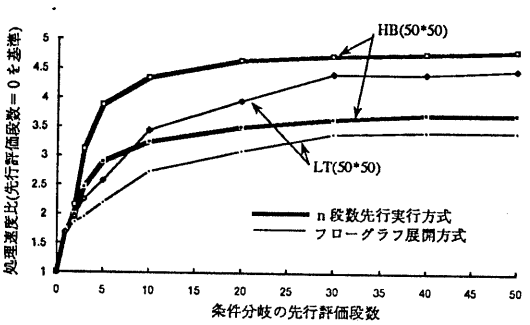


図５　ｎ段先行実行方式の処理速度向上比

３．ｼﾝｸﾞﾙﾌﾟﾛｾｯｻでの先行評価実現上の問題

本節では，シングルプロセッサ上（例：ＶＬＩＷ計算機）でｎ段先行評価を実現する場合の問題点を明かにする．

３．１ 対象Boolean Recurrenceﾙｰﾌﾟ

対象とするBoolean Recurrenceﾙｰﾌﾟは，

(1)唯一の制御の戻り点と，ループ内部に１つ以上の条件分岐文を持ち，

(2)条件分岐を決定する真理値へのフロー依存を持つデータが繰り返し間でサイクリックな依存を持つループである．ただし，ここで言うサイクリックとは，ループ内の全ての条件分岐文からのコントロール依存を各条件分岐文毎に１本以上（true側あるいはfalse側を通り）もち，かつ，その依存がループ間でサイクリックな関係を持つことを示す．簡単に言うと，あるパスを選択するとサイクリックなデータ依存関係が繰り返し間で発生するが，ある別のパスを選択するとサイクリックなデータ依存が変わるかあるいはなくなるような依存関係を持つことを意味する．具体的には，N個の条件分岐の真理値データ{B1,B2,..BN}が存在した時，それぞれの真理値データに対してフロー依存を持つ依存サイクル（データ依存とコントロール依存で構成）{Cy1,Cy2,...,Cyn}が存在する時，ある２つのサイクル間について，Cya->Cyb->Cyaのフロー依存が存在するサイクルを１つにまとめた時，最終的に{Cy1,Cy2,...,Cyn}が１つのサイクルになり，かつ，その中に{B1,B2,..BN}からのコントロール依存を各Biから１本以上持つような場合を言う．この時，複数の独立なサイクルができた場合には，それら２つのサイクル間で依存がないため，独立に計算を進めることができる．

対象ループの例を図６に示す．図６の例では，S1->S2->S5->S6->S8->S9->S10->S11->S15->S1の順で依存サイクルを構成しており，S2->S5，S11->S15のコントロール依存を含んでいる．従って，繰り返し間でBoolean Recurrenceを持ち，並列に実行することができない．ここで，S5->S8及びS15->S1へのデータ依存が無く，代わりにS7->S1へのデータ依存が存在したと仮定すると，本ループは２つの独立なデータ依存サイクルから構成されることになり，{S1,S2,S5,S6,S7}の計算を各繰り返しに対してまず行い，その後残りの計算を行えばよい．この場合，２つのループに分割でき，{S1,S2,S5,S6,S7}と{S3,S4,S8,S9,S10,S11,S12,S13,S14,S15}から構成される２つのBoolean Recurrenceループとなる．

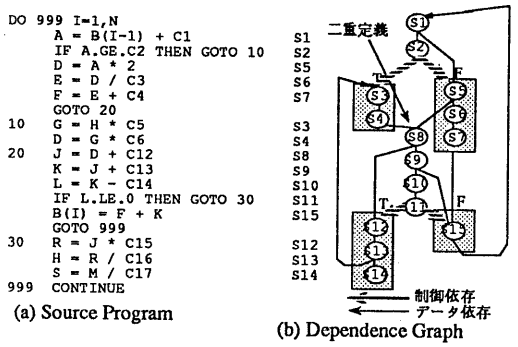


図６　Boolean Recurrenceループの例

３．２ ディレイ計算グラフ

シングルプロセッサ（例：ＶＬＩＷ計算機）上でｎ段先行実行を行う場合には，各基本ブロック内のデータ依存で接続されたパス毎（基本パスと呼ぶ）に最速の実行開始タイミングを求めなくてはならない．しかし，Boolean Recurrenceの存在するループでは，過去の条件分岐選択経路によって最速の実行タイミングが動的に決定される．これは，図７に示すようなディレイ計算グラフを考えればわかりやすい．図７は図６のディレイ計算グラフであり，各ノードが基本パス（本例の場合には，各基本ブロックに１本のデータパスしか存在しないため，基本パス＝基本ブロックとなっている）に対応する．またアークは，データ依存があることを示し，横に書かれた数字がソースノード実行後のディスティネーションノード実行までのディレイ時間を表す．

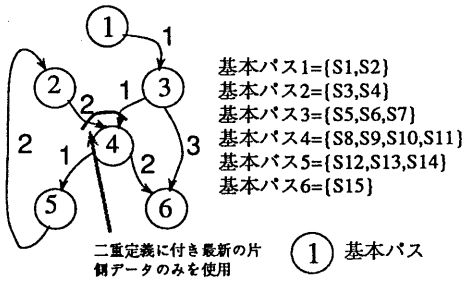


図７　ディレイ計算グラフ

例えば，基本パス３は基本パス１実行後，１単位時間後に実行開始できることを示す．アークが存在しない基本パス間はデータ依存がないために独立に実行を開始することができる．例えば，基本パス１と基本パス２の間にはアークが存在しない．従って，基本パス２は過去に実行した基本パス５の実行開始時間＋２の時刻までさかのぼって実行を開始できる．また，基本パス４では基本パス３から制御が移行した場合にはディレイ１，基本パス２から制御が移行した場合にはディレイ２で実行開始できる．これに対して，基本パス６は基本パス３と４の両方からディレイを受ける．

３．３ ｼﾝｸﾞﾙﾌﾟﾛｾｯｻによる実現上の問題

ディレイ計算グラフを用いることによって，各基本パスの最速実行タイミングを求めることができる．しかし，各基本パスの最速実行タイミングは過去の条件分岐選択経路に依存しているため，コンパイル時に静的にプログラムを作成することができない．従って，ｎ段分の先行実行を行うコードを予め作成しておき，ｎ段毎に同期をとりながら実行しなければならない．また，ＶＬＩＷ計算機では，多くても２０台程度の演算器しか持っていないために，各ステップを２０演算以内に抑えるためのスケジューリングも必要となる．

４．データ駆動方式を用いた先行評価

本節では，データ駆動方式を用いたｎ段先行実行方式を提案する．データ駆動方式を用いることの利点は，最速の実行タイミングを静的に求める必要がなく，データ駆動原理に基づいて実行時に自動的に最速の実行タイミングで実行開始できる点にある．

４．１ 概要

従来我々が提案しているフローグラフ展開や，3節で述べたシングルプロセッサによる方法では，n段の先行実行毎に同期をとる必要がある．これに対して，データ駆動方式を用いた方法では，データが利用可能となった時点で演算が発火し実行されるため，実行タイミングの制御が必要なくなり，プログラムの再利用が可能となる．プログラムの再利用を行うために，n段先行実行ではカラーを用いて各段の実行を区別する．図8に実行の概要を示す．図8に示すように，各段毎にゲート命令を挿入することによって，n段先行実行の制御を行う．また，同一プログラムに対して，各段毎の実行を区別するためにカラーを用いる．またカラーを再利用しカラーのオーバーフローを避ける．さらに，先行実行中の副作用を避けるために，共有メモリへのストア等の制限，及び実行時エラーの回避を行う．以下では，それぞれの解決手法を示す．

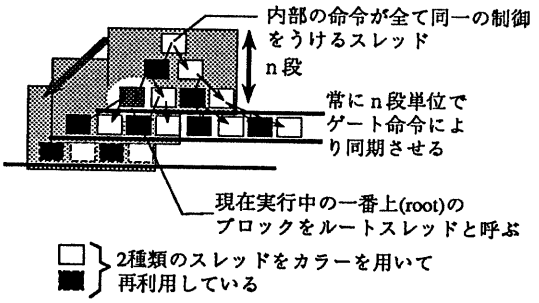
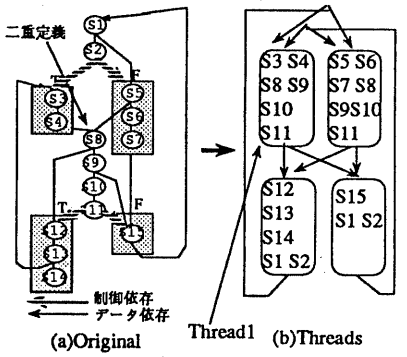


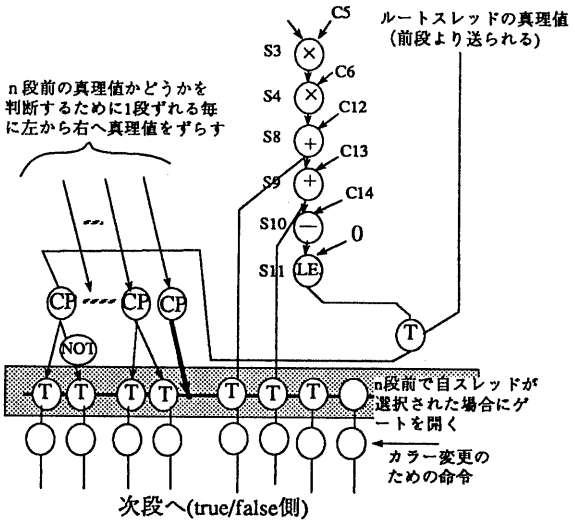
図8　n段先行実行の概要

4.2 コード生成手法

以下では，図6に示した例を用いて説明を進める．まず，図6のプログラム中，同一の制御を受ける基本パス同志を融合し，図9(b)に示す4つのスレッドを作成する．以下の処理は全てのスレッドについて共通の処理であるので，ここではスレッド1を例にとって説明する．

常にn段先行実行を行うために，スレッドからの出力データに対してTFゲートを挿入する．このTFゲートへの真理値はn段前のブロックから送られてくる．この時，図9(c)に変換例を示す．図中では，n段前から送られてきたことを確認するために，各段を通過する毎にデータのポジションをずらしている．また，このn段前からの真理値は，先行評価によって生成されたものではなく，正しい真理値であることを保証しなくてはならない．従って，図8におけるn段先行実行時のルートスレッドであることを保証するために，ルートスレッドを示すデータを用いている．また，前段からの真理値を次段へ送るため，及び，自分のスレッドで定義したデータを次段へ送るためのT(True)ゲートも示されている．





(c)変換コード概略

図9　n段先行実行制御コード

4.3 カラー再利用手法

同一スレッドに対する各段の実行を区別するためにカラーを用いる．しかし，データ駆動計算機におけるカラー資源は有限であるため，カラーを再使用する必要がある．このため，カラーの回収及び再割当を以下の手順で行う．

(1)各スレッドに対して，n段先行実行した時に同時に起動される最大の世代数を予め求めておき，その世代数分のカラーを初期カラーとして投入する．

(2)カラーの回収は，スレッド単位で次の条件が満たされた場合に行う．

a.スレッドから全てのデータが出力された時．

b.スレッドにおいて全てのデータが回収された時．

a.の条件は，該当スレッドがn段先行実行中の状態にあり，バイパスデータが全て次段へ流れた場合であり，b.の条件は，n段前の条件判定の結果，自スレッドが選択されないことが確定した場合である．

(3)カラーの回収後の再割当を次の手順で行う．

a.回収されたカラー番号をデータへ変換し(CGET:Color GET命令)カラースプール部分(図10)へ送る．

b.カラースプール部分では，n段前の条件分岐条件がtrueで流れて来た場合に限り，次段へ送るデータに対して新たなカラーを割り付ける(CSET:Color Set命令)．この時，図10中の※の演算の左入力にはカラースプールされているカラー0の値を持ったデータが多数存在する．この時，右入力にある1つのデータが入力されるとカラースプールされている多数のデータ中1つのデータとマッチングする．一晴一では，このようなマッチング形態を許している．

以上の手順によってカラー再利用コードを構成した図を図10に示す．

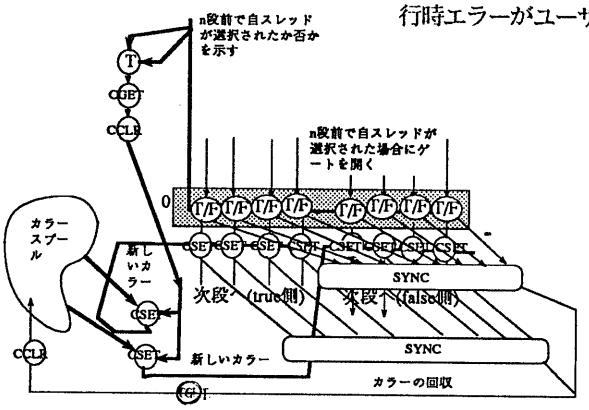


図10　カラーの回収再割当

4.4 副作用回避手法

n段先行実行時には，共有メモリからの読み出しのみを許し，書き込みを禁止する．すなわち，自スレッドにおける条件が決定した時点で書き込みを行う．このため，図11に示すように，ルートスレッド情報（4.2参照）の真理値が真であるときのみストアするようなコードを生成する．（図11）

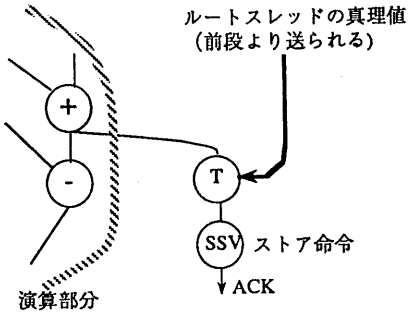


図11 副作用回避手法

4.5 実行時エラー回避手法

n段先行実行では，制御依存とデータ依存を分離して実行を行うため，実行時に0除算等のエラーが発生する可能性がある．このような実行時エラーは[HKYM89]に示すように，エラートークンを用いて処理する．すなわち，n段先行実行によって発生した実行時エラーは，n段先行実行中，エラートークンとして保持する．そして，各段の制御が正式に決定された段階で，ゲート命令により消去あるいは，実行時エラーがユーザに報告される．

4.6 今後の改善点

本節で示した手法により，n段先行実行をカラーを用いて行うことが可能となる．しかし，新たに発生する問題として，n段先行評価によって生じる制御コードの増加によるオーバヘッドが挙げられる．このため，制御コード増加によるオーバヘッドを隠すために，スレッド本体をある程度大きくする必要がある．

また，n段先行実行は，ここに示した各種手法を用いることによって，カラーの代わりに関数起動によって実現することも可能であるので，その手法についても考察を進める予定である．

さらに，n段先先行実行によって，多数のプロセッサを使用する必要がある．この問題に対しては，データ依存関係からそれぞれのパスに応じて最大の先行評価段数を概算することができるので，今後，無駄なパスについてはn段よりも小さい段数での実行を行っていけるような手法に拡張したいと考えている．

5. むすび

本報告では，Boolean Recurrenceを持つループについて，条件分岐を常にn段先行実行する方式をデータ駆動計算機上で実現するための手法を示した．今後は，本稿で述べた方式を定式化していくと共に，データ駆動計算機のシミュレータを用いて，速度向上率と使用プロセッサの割合について評価を進めて行きたいと考えている．

謝辞

本稿作成にあたり，討議いただいた村岡研究室諸氏に感謝いたします．

参考文献

[BaGa84] Utpal Banerjee, D.D.Gajski:"Fast Execution of Loop with IF statements",IEEE Trans. on Computers,Vol.C-33,pp.1030-1033,1984

[Cytr86] Ron Cytron:"Doacross:Beyond Vectrization for Multiprocessors", Proc. of ICPP'86, pp. 836-844, 1986

[HKYM89]萩本,草野,山名,村岡:"並列処理システム－晴－における実行時エラーの処理",情処第39回全大, 6W-5,1989

[RiFo72] E.M.Riseman,C.C.Foster:"The Inhibitation of Potential Parallelism by Conditional Jumps", IEEE Trans. on Computers, pp.1405-1411, 1972

[Uht88] A.K.Uht:"Requirements for Optimal 'xecution of Loops with Tests", Proc. of ICS88, pp. 230-237, 1988

[YMHM88]H.Yamana,T.Marushima,T.Hagiwara,Y.Muraoka:"System Architecture of the `Parallel Processing System -Harray-", Proc. of ICS'88, pp.76-89, 1988

[YYKM91] 山名,安江,神舘,村岡:"並列処理システム－晴－におけるフローグラフ展開を用いた条件分岐の並列実行",早稲田大学情報科学研究教育センター紀要,Vol.12,pp.8-18,1991