計算機アーキテクチャ 75－1
（1989. 3. 23）

並列処理システム－晴－における フローグラフ展開の評価

萩原孝　山名早人　神舘純　村岡洋一
早稲田大学　理工学部

本報告では，並列処理システム－晴－上でFORTRANプログラムを高速実行するためのフローグラフ展開による仮実行方式を提案し評価する．

FORTRANプログラムをそのままデータフローグラフに変換した場合，（1）単一代入則の欠如，（2）明示的な制御フローの存在，といった理由から多数の制御ゲートが必要となり，プログラムの並列性が制限される．そこで，これらの制御ゲートを排除したフローグラフ展開の方法を提案する．－晴－のソフトウェア・シミュレータにより，いくつかのプログラムについて評価したところ，制御が決定してから実行を開始する先行実行方式に比較して約1.5倍の処理性能向上が見込まれることがわかった．今回，評価に用いたプログラムは，小規模なものであり，制御ゲート数が数十と少ない．しかし，プログラムが大規模になった場合には，制御ゲート数が増し，本方式はさらに有利になると考えられる．

Evaluation of Unfolded Flow Graph for the Parallel Processing System -Harray-

Takashi Hagiwara, Hayato Yamana, Jun Kohdate, and Yoichi Muraoka

School of Science and Engineering, Waseda University
3-4-1, Okubo, Shinjuku-ku, Tokyo, 169 JAPAN
BITNET/JUNET MURA001@JPNWAS00 / harray@muraok.waseda.junet

The perpase of this paper is to propose and evaluate the preceding activation scheme with unfolded flow graph.

The problem in restructuring a FORTRAN program into dataflow graph is that (1) a FORTRAN program does not written in single assignment rule and (2) it has explicit control flow. This problem generates small parallelism because many gate-operations are used to synchronize data. Therefore, reducing these gate-operation is the key to express parallelism from a FORTRAN program. In this paper, an unfolding scheme of flow graph is proposed to reduce these gate-operations. Then, the evaluation by software simulation with small scale programs is presented. It shows that the execution speed with the scheme is about 1.5 times as fast as that with the extended activation scheme which initiates the execution when it is confirmed the basic-block will be selected at conditional branch. The scheme will be more effective than the extended activation scheme when large scale program is simulated because it includes many gate-operations.

１．はじめに

我々は，科学技術計算の高並列処理を目的とした並列処理システム－晴－を提案している［１］［２］．－晴－は，科学技術計算の分野で膨大なソフトウェア資産を持つFORTRANを対象に，１０００台以上のデータフロー方式のプロセッサを使って演算レベルの並列処理を実現することを目指したマルチプロセッサシステムである．－晴－の特徴は，

・プログラムをマクロブロックと呼ぶタスク群に分割し，マクロブロック間はFORTRANなどの手続き型言語の特徴であるコントロールフローにより実行制御．

・マクロブロック内は演算（命令）レベルの並列処理を行うためにデータフロー方式により実行制御．

という上位（マクロ）レベルでのコントロールフロー実行と下位（ミクロ）レベルでのデータフロー実行の階層的な実行制御方式を取り入れていることである．そして，この二階層の実行制御方式を採用することで－晴－は，コントロールフロー計算機（ノイマン型計算機）を前提としているFORTRANで記述されたプログラムをデータフロー実行する場合のセマンティック・ギャップの解消とデータフロー計算機の問題点である並列性の制御を可能としている．

これまで，我々は－晴－のアーキテクチャの特徴を活用し，既存のFORTRANプログラムの高速処理を実現するために，同期実行，先行実行，仮実行方式の３種類のマクロブロックの実行（起動）方式を提案している［３］．本稿では，これらの実行方式のうち，特に高速化が期待される仮実行方式について，その実現のためのフローグラフ展開と呼ぶ一種のプログラム変換技法を提案し，ソフトウエア・シミュレータによる評価を行ったので，その結果を報告する．

以下，第２節では，－晴－のアーキテクチャと３種類の実行方式について説明する．第３節では，FORTRANプログラムをデータフロー計算機である－晴－上で実行する場合の問題点を検討する．第４節では，３節で指摘した問題を解決し，仮実行方式を実現するためのフローグラフ展開を提案する．第５節では，フローグラフ展開による仮実行方式の評価を示す．

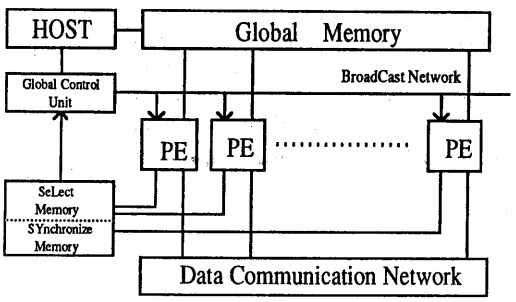


図１　－晴－の全体構成

２．－晴－のアーキテクチャと実行方式

２．１－晴－のアーキテクチャ

－晴－の全体構成を図１に示す．ＧＣＵ（Global Control Unit）が，システム全体の実行管理をする．すなわちマクロブロックを実行単位とするＧＣＧ（Global Control Graph）に従って個々のマクロブロックを並列に実行させる（図２）．ＢＣＮ（Broad Cast Network）は，ＧＣＵがＰＥに対してマクロブロックの実行などの指示を伝えるための放送ネットワークである．ＰＥ（Processing Element）は，実際の計算処理を行う．ＰＥ内部は，複数の演算器を備えたデータフローリング構造で，演算レベルの並列処理が実現される．ＧＭ（Global Memory）は，配列などの構造データを格納しておく共有メモリで，マルチアクセス構成を想定している．ＤＣＮ（Data Communication Network）は，ＰＥ間のデータ交換のための通信ネットワークである．ＳＹＭ（SYnchronize Memory）は，複数のＰＥで実行されるマクロブロックの終了を高速に検出するためのユニットである．ＳＬＭ（SeLect Memory）は，ＰＥが条件分岐の計算結果を報告するためのユニットである．

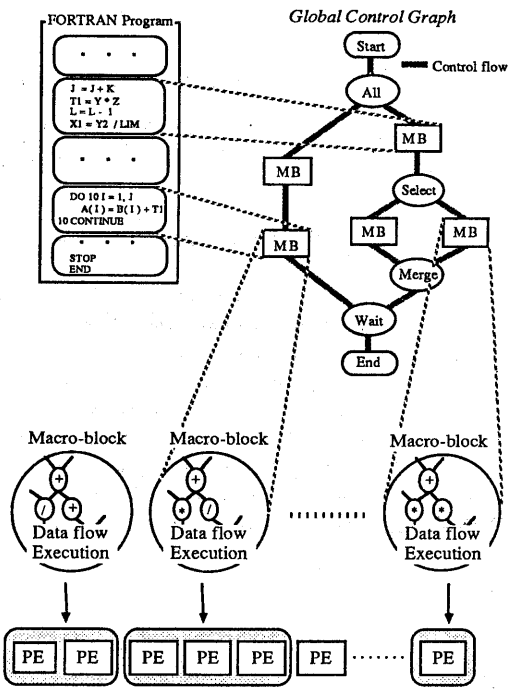


図２　Global Control Graphによる実行制御

２．２　マクロブロックの実行方式

プログラムの並列性を制限している要因の一つは，アルゴリズムに固有の依存性である．この依存性は，プログラム（言語）レベルでは２つの依存性に分けられる．一つは，*データ依存性*（オペランドが有効になるまで命令の実行ができないという依存性）であり，もう一つは，*制御依存性*（条件付き分岐文などで条件が決定するまで分岐先の命令を実行できないという依存性）である．

Risemanら［４］は，一般的なプログラムでは，データ依存性の量は少なく，このため多量の並列性が得られるが，条件付き分岐のような制御依存性は非常に多く，並列実行の可能性を制限していることを実際のプログラムを対象にしたシミュレーションにより確認している．

我々は，この２種類の依存性を考慮して，マクロブロックの実行（起動）方式として，次の３つの方式を提案している（表１）．

（１）同期実行（起動）
　ＩＦ文等の条件分岐により必ず実行されることが決定し，かつそのマクロブロックに必要なデータがすべて揃ってから実行を開始する方式

（２）先行実行（起動）
　必要な全データの到着を待たずに，実行することが確定した段階で実行を開始する方式

（３）仮実行（起動）
　条件分岐によって，実行が決定されない段階で，実行を開始する方式

表１　マクロブロック実行起動方式

起動呼称		制御条件	データ条件
本実行	同期実行	必ず実行されることが決定している	全てのデータが到着済み
	先行実行		到着しているかは不確定
仮実行		実行されるかは未定	

これらのマクロブロックの実行方式について，既存の科学技術計算プログラムを使って，ステートメントレベルで比較するシミュレーションを行った結果，表２のような結果を得ている［３］．このシミュレーションにおいて，先行実行方式がちょうど制御依存性によって並列性を制限された場合に当たり，仮実行方式が制御依存性を無視した場合に当たる．これらの結果から，プログラムから制御依存性を除くことができれば，既存の逐次処理用のプログラムでもかなり並列性を引き出すことが可能であり，高速化が見込まれることがわかる．

表２　起動方式間の速度比

起動方式	速 度 比
同期実行	1.00
先行実行	1.11
仮実行	3.68

３．FORTRAN言語の問題点

プログラミング言語（特にFORTRAN）のレベルで見ると，制御依存性を規定しているのは，ＩＦ文などの条件分岐文である．具体的には，この制御依存性は条件分岐により「次に実行するコードブロック（基本ブロック）の決定」という形で実現されている．一晴一では既存プログラムのコントロールフローを上位レベルの実行制御に用いるため，基本的にマクロブロックとして制御依存性の単位である基本ブロックを選択している．しかしながら，元来フォン・ノイマン型のアーキテクチャを前提にしているFORTRAN（手続き型言語）で記述されたプログラムをデータフロー計算機上で実行するには，その間のセマンティックギャップが大きい．本節では，FORTRANプログラムをデータフロー計算機上で実行する際の問題点を挙げ，後述する「フローグラフ展開」と呼ぶプログラム変換技法の必要性を示す．

これまでデータフロー計算機の言語としては，ＶＡＬ［５］，Ｉｄ［６］，ＤＦＣ［７］などが提案されている．これらの言語はデータフローグラフとの対応を考慮して設計されている．しかし，FORTRANのようなデータフローグラフとの対応を考慮していない手続き型言語を対象とした場合，手続き型言語をデータフローグラフに変換する上で次の２点が問題となる．

（１）データフロー言語では，基本的に単一代入則が適用されているが，FORTRANではこれが適応されていない．

（２）FORTRANには，GOTO文のように明示的な制御フローの文が存在する．

ここで言う単一代入則とは，変数への値の代入は一度限りしか許さないという規則である．

（１）の問題は，記憶セルの概念に基づいた代入の弊害を引き起こす．ノイマン型計算機では代入により，記憶セルの内容を時々刻々変更することによって計算が進められる．別な言い方をすれば，ノイマン型計算機は状態を変化させることで（所謂副作用によって）計算を進めている．一方，データフロー計算機では，元々必要なオペランド（値）が到着すると演算を行い，そして結果を出すという関数的な概念に基づいている．したがって，副作用を基本としたFORTRANなどの手続き型言語で記述されたプログラムをデータフロー計算機で実行する場合，

その状態の概念をデータフローグラフとして表さなければならない．具体的には，このことはデータフローグラフ上の制御ノードの増加という形で現われる［８］．例えば，図３に示すように複数箇所で定義される値を参照する可能性が生じた場合，制御ゲートを設けてデータの選択を行なわなければならないわけである．

（２）の問題は，さらに上記の問題を拡大させる．条件付きGOTO文などの制御を移行する文がプログラムに含まれた場合（図４），データの選択のために制御ゲートが一段と増加する．また，これらの制御ゲートは，データ選択のための一種の同期点となり，データフロー実行の非同期性という利点を失わせることにもつながる．

このようにFORTRANには，単一代入則を実現していないことに加えてGOTO文などの制御の移行を指示する文が存在する．このため，FORTRANプログラムをこのままデータフローグラフに変換して実行した場合，多くの制御ゲートによって高速化が妨げられることが予想される．

また，これらのデータを選択するための制御ゲートがデータフローグラフ上に表されることは－晴－の上位レベルのコントロールフローの利点が薄れることになる．すなわち，－晴－は上位レベルにおいてコントロールフローによる実行制御を可能にしているにも関わらず，下位レベルでも「データパケットの選択」と言う形で制御依存性を反映することが必要となり，二重に制御依存性を保持することになる．このため，マクロブロック単位の実行制御が生かされないことになる．

これらの問題点，

（ａ）手続き型言語特有の副作用による計算実行の概念を反映して，FORTRANプログラムをデータフローグラフに変換した場合，データ選択のための制御ゲートが増加し，高速化を妨げる可能性が高い．

（ｂ）（ａ）によるデータフローグラフ上への制御ゲートの頻出によって，－晴－の二階層の実行制御機構が生かされない．

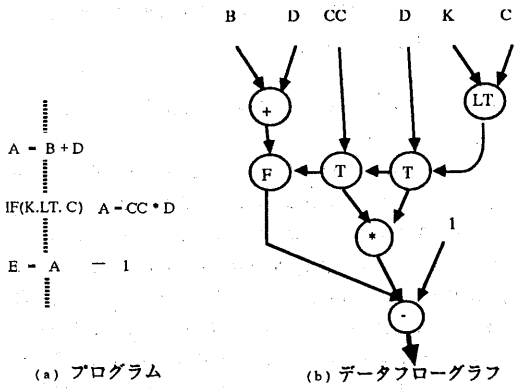


図３　FORTRAN言語の問題点(1)

の原因は，FORTRANプログラムが持つ，制御構造（すなわち，フローグラフの形状）をそのままデータフローグラフ中に実現しようとしたことにある．これまでParafrase［９］に代表されるプログラムの並列性を高めるための並列化コンパイラでは，主にデータ依存性を中心にプログラム解析を行い，冗長なデータ依存性を除去するプログラム変換を行って来た．もちろん，局所的には制御依存性の除去も行われて来たが，基本的にノイマン型計算機上での実行を前提としていたため，その制御構造は保持されて来た．

しかし，－晴－では，データフロー実行を基本としており，先に述べたように既存プログラムにおいては，制御依存性がプログラムの並列性を大きく制限していることから，－晴－で既存プログラムの高速化を図るためにはプログラムの制御構造の変換，すなわち（制御）フローグラフの変換が必須である．

４．フローグラフ展開による仮実行方式の実現

２節で述べたように制御フローを中心に記述された既存プログラムは，データ依存性に加えて，条件分岐のような制御依存性によってその並列性が制限されている．仮実行方式は，－晴－の階層化された実行制御方式の特徴を生かして，データ依存性と制御依存性を分離して既存プログラムの高速化を図る実行方式である．本節では，この仮実行方式を実現するための「フローグラフ展開」技法を提案し，あわせて－晴－上での仮実行方式の実現方法を述べる．

４．１　フローグラフ展開の適用範囲

プログラムの並列性はDOALLループのような高い並列性を持つ部分とそうでない部分に分れる．１０００台以上のプロセッサを持つ－晴－では，並列性の低い部分ではほとんどのプロセッサがIdle状態となる．仮実行方式はこれらのIdle状態のプロセッサを使って，最終的には無効となる可能性のあるデータでも先行計算しておき，プログラムの高速化を実現することを目的

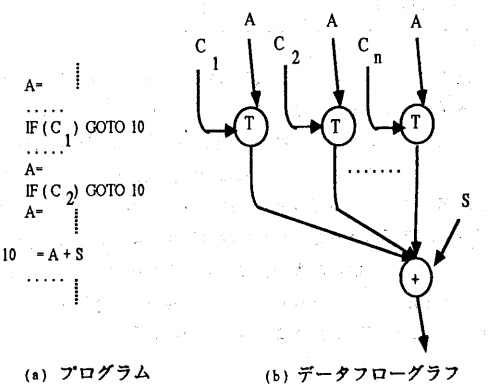


図４　FORTRAN言語の問題点(2)

にしている（図５）．したがって，まずフローグラフ展開の適用範囲は，DOALLループのような高並列処理可能な部分を除いた部分を対象とする．また，フローグラフ展開は，プログラム中の制御依存性を除去することで並列性を高めることを目的としているので，条件付き分岐文などで細かく分割されているスカラ演算部分や制御依存性を含んだＤＯループ，そして後方分岐文（Backward GOTO statement）などによって形成されるループ部分が主な対象部分と成りえる．

ただし，ループ部分については，ループ回数が動的に決定される場合も多く，展開回数の決定が困難である．このため，フローグラフ展開するよりも，むしろDOACROSSループのようにＰＥ間でパイプライン実行を行ったほうが高速化につながる可能性もある．そこで，フローグラフ展開は，DOALLループのような並列性の高い部分を含まず，かつループを形成している後方分岐文などを越えない部分を基本的な展開対象範囲とする．

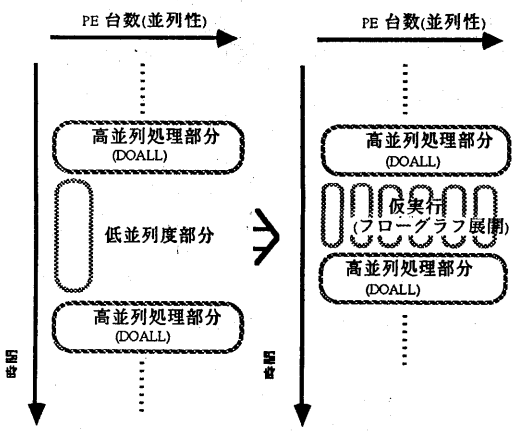


図５　フローグラフ展開の適用範囲

４．２　フローグラフ展開の手順のその展開例

データ選択の制御ゲートを必要とする原因のひとつは単一代入則を適用していないことであるが，これをプログラムの制御構造の面から見るとフローグラフ上で制御の合流が起こるためである．つまり，単一代入則を適用していなくても，プログラムの制御フローの経路が固定されていれば，データを選択する必要はない．そこで，フローグラフ展開の手順の（１）として，データ選択の制御ゲートを消去するために，フローグラフ上の制御の合流を除去する次の規則をフローグラフに適用する．これを（フローグラフの）『下方展開』と呼ぶ．

規則（１）フローグラフ上で制御フローが合流する場合，その合流点以後のフローグラフを複製し，制御の合流を除去する．

フローグラフの下方展開の例を図６に示す．図６（ａ）は，展開前のフローグラフである．同図では，制御フローの合流があり，変数Ａが複数の基本ブロック（以後，ＢＢとする）で定義されている．そのため，このままデータフロー実行するとＢＢ４，ＢＢ６，ＢＢ８は，変数Ａの選択のために制御ゲートを必要とする．例えば，ＢＢ４で参照される変数ＡはＢＢ１とＢＢ３で定義される可能性があり，C_1とC_3によるゲートを設けなければならない．同図（ｂ）は制御フローが合流するＢＢ４以下とＢＢ８をコピーして，下方展開したものである．

この下方展開によって各制御フローに沿った変数の定義・参照関係は一意に定まり，単一代入則の問題は解決される．また，各データフローグラフ内にはデータ選択用の制御ゲートが必要なくなるため，本質的な計算部分のみとなる．このように下方展開した状態で，全てのブロックを仮実行で起動し，制御が決定した段階で本実行への移行，あるいは実行の無効化をすることにより，制御ゲートを排除した仮実行が実現できる．

しかし，下方展開のみでは，次に挙げる点が問題として残る．

・展開された範囲の制御フローのうち，最終的に有効となるフローは１つのみであるが，他フローにおけるデータ転送も仮実行中に行なわれる．このため，本当に必要であるデータの転送（最終的に有効であるフローに属するデータ）が，他の必要でなかったデータの転送（最終的に有効でないフローに属するデータ）により遅延を受ける．

・仮実行から本実行への移行および，制御がそのブロックへ移らなかった時の仮実行無効化の制御がGCUに集中し，過負荷となる．

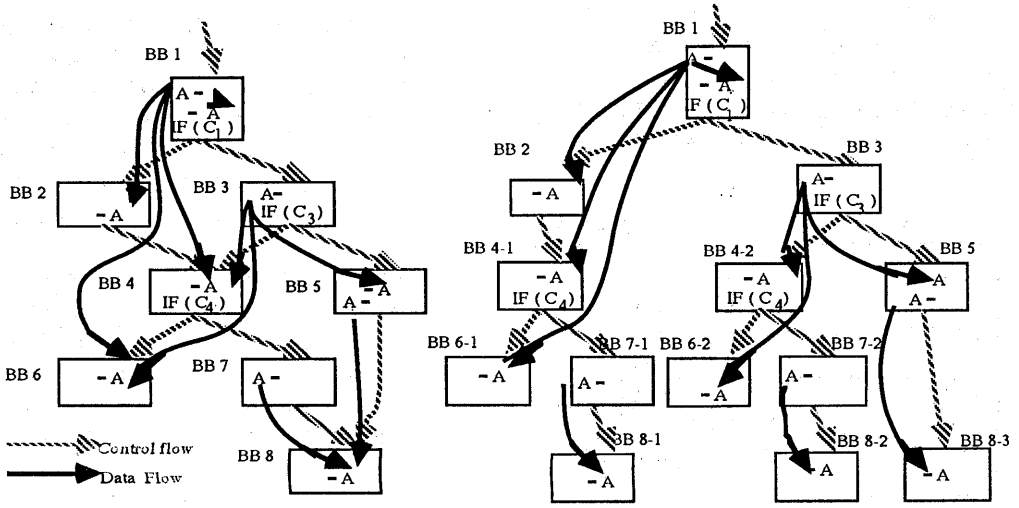
これらの問題点を解決するため，下方展開後にさらに次の規則（２）および規則（３）を適用し，フローグラフの分岐部分に対して上方に展開し（同図（ｃ）），これらの問題を解決する．

規則（２）フローグラフ上で制御フローの分岐が存在するとき，その分岐点以前のフローグラフを複製して，制御フローの分岐を削除する．この時，フローグラフがちょうど簾状になることから，これを『スレッド』と呼ぶことにする．

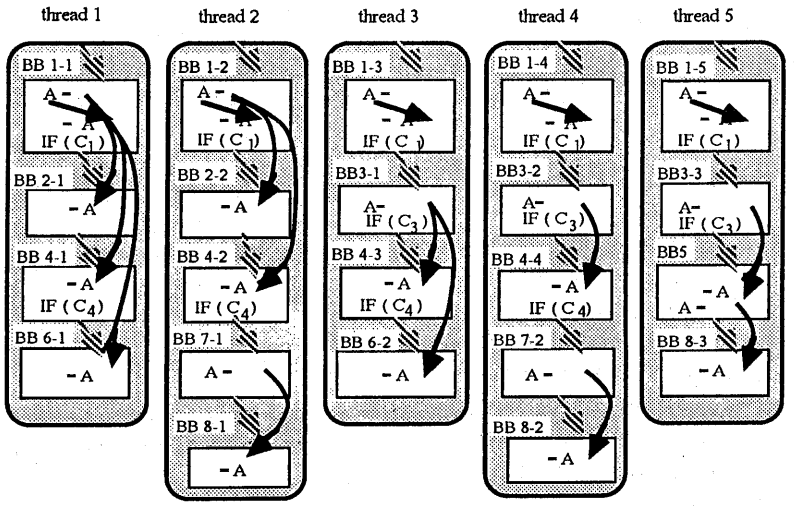
規則（３）展開された各制御フローごとのブロックをまとめてひとつのマクロブロックとする．

上方に展開すると，フローグラフはちょうど簾状になる．このように簾状に制御フローを完全に分離すると，もはやスレッド間の通信は存在しない．また，各スレッド毎で自分のスレッドが有効であるか無効であるかを判断できるため，スレッド間

の制御依存もなくなる．したがって，これらのスレッドをそれぞれＰＥ（群）に割り付けて実行すれば，他に影響されることなく独立して計算が行える．そして，規則（３）によりこの際状になったスレッドをひとつのマクロブロックとしてまとめ，新たに上位レベルの制御単位と考えれば，ＧＣＵはこれらのスレッドを一つのマクロブロックとして同時に起動でき，終了報告もそれぞれのスレッドから実行の有効・無効の報告を受け，後処理をするだけで良く，ＧＣＵの負荷を軽減できる．



（ａ）フローグラフ　　（ｂ）フローグラフの下方展開



（ｃ）フローグラフの上方展開

図６　フローグラフの展開例

５．評価

フローグラフ展開による仮実行方式の評価を一晴一のソフトウェア・シミュレータ［2］を用いて行なった．使用したプログラムは，複数の科学技術計算プログラムの中から，フローグラフ展開の対象となる部分を抜き出したものであり，表３に示すような特長を持つ．

評価項目として，以下の３種類の方式で実行したときの実行時間を調べ比較した．

（１）先行実行方式
　　プログラム中の各基本ブロックをマクロブロックとし，制御が決定した段階で，各マクロブロックの実行を開始する方式．

（２）１マクロブロック方式
　　全体を１マクロブロックとして実行する方式．

（３）フローグラフ展開による仮実行方式
　　プログラムから制御ゲートを取り除き，各スレッドをマクロブロックの単位として実行する方式．

図７に先行実行方式を基準としたときの，各方式の実行速度向上率を示す．これらのプログラムでは，条件分岐によって実行時間が一意に定まらないため，比較に当たっては，全ての条件分岐の組み合わせで実行した結果の平均を用いている．図７より，フローグラフ展開による仮実行方式を用いた場合，先行実行方式に比較して約1.5倍，１マクロブロック方式に比較して約1.2倍の処理性能向上（いずれも３つのプログラムの平均）が見込まれることがわかる．

先行実行方式は，制御が決定した段階でマクロブロックの実行を開始する方式である．これは，関数型言語で言えば，全てのオペランドが揃うのを待たずに関数が必要となった時点で実行を開始させる，所謂先行評価に当たる．これに対して，１マ

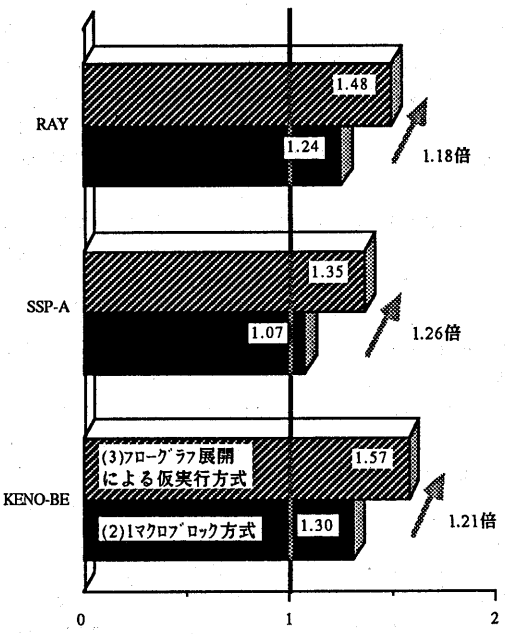
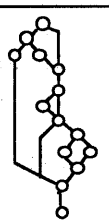
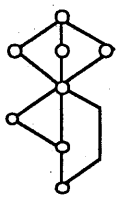
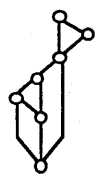


図７　先行実行方式を基準とした時の
各実行方式の実行速度向上率

表3　評価対象プログラム

プログラム名	KENO-BE	SSP-A	RAY
プログラムの説明	原子力コードKENO-IVのメインルーチンの一部	Scientific Subroutine Packageの一部	レイトレーシングプログラムの一部
全ノード数	143	159	222
制御ゲート数	15	16	15
基本ブロック数	14	8	7
スレッド数	13	9	8
フローグラフ ○ は,基本ブロックを表す			

クロブロック方式やフローグラフ展開による仮実行方式は，制御が決定しない段階で，計算可能な部分を出来るだけ先に計算してしまう方式である．このように，必ずしも必要でないかも知れない部分を先に計算することによって，先行実行方式に比較して約1.2倍の処理性能向上が見込まれることがわかった．

１マクロブロック方式では，制御ゲートを用いてデータの流れを制御している．これに対して，フローグラフ展開による仮実行方式では，プログラムを各制御フロー毎にスレッドに分割することによって制御ゲートをなくしている．このため，フローグラフ展開による仮実行方式では，１マクロブロック方式に比較して，制御ゲートが無くなった分，約1.2倍処理性能が向上した．

今回，評価に用いたプログラムは，ノード数が多くても200個という小規模なプログラムである．このため，データの流れを制御する制御ゲート数が十数個と少ない．しかし，プログラムが大規模になると，データ依存関係が増大し，さらに制御ゲート数が増えると考えられる．また，１マクロブロック方式のように１つのマクロブロックとして実行すると，条件分岐によって最終的には必要とされない演算（無効演算）と，最終的に必要となる演算（有効演算）が区別されず実行される．このため，有効演算が無効演算によって実行の遅延を受けると考えられる．この遅延は，プログラムが大規模になった場合に顕著に現われる．

これらの問題は，フローグラフ展開による仮実行方式により解決される．即ち，フローグラフによる仮実行方式では，制御ゲートを排除し，かつ，１つのスレッドを１つのマクロブロックとして実行するため，有効演算が無効演算によって実行の遅延を受けることがない．このため，プログラムが大規模になった場合，フローグラフ展開による仮実行方式は，１マクロブロック方式に比較してさらに有利になると考えられる．

６．　まとめ

本稿では，並列処理システム－晴－におけるFORTRANプログラムの高速処理を実現するための仮実行方式について検討し，この実行方式を効率よく実現するためのフローグラフ展開技法を提案・評価した．いくつかの簡単なプログラムについてシミュレーション評価した結果，フローグラフ展開による仮実行方式により，制御が決定してから実行を開始する先行実行方式に比較して約1.5倍の処理性能向上が見込まれることがわかった．今回，評価に用いたプログラムは，ノード数が多くても200個という小規模なプログラムである．このため，データの流れを制御する制御ゲート数が十数個と少ない．しかし，プログラムが大規模になると，データ依存関係が増大し，さらに制御ゲート数が増えると考えられるため，フローグラフ展開による仮実行方式は，さらに有利になると考えられる．

このようにフローグラフ展開することによって個々のスレッドはオプティマイズされる．しかし，コードの複製を行なうためマクロブロック全体としてコード量が増え，他のマクロブロックからの参照データ数が増加する．これは，－晴－のネットワークであるDCNのデータ転送能力が高くなければならないことを意味している．この点について，今回は評価を行わなかったが，各スレッドが必要とするデータは，そのほとんどが共通であるため，DCNの通信方式として放送機能を備えることにより解決できると思われる．

今後は，プログラム全体に対してシミュレーションを行ない，DCNの構成やデータ転送能力について検討していく予定である．

謝辞

本研究遂行にあたり，御支援いただいた本研究室の草野義博氏，安江俊明氏，樋口和広氏に感謝いたします．

参考文献

[1] H.Yamana,T.Marushima,et al.:"System Architecture of Parallel Processing System -Harray-", Proc. of Int. Conf. on Supercomputing, pp.76-89 (1988)

[2] 萩原,山名,丸島,村岡:"並列処理システム-晴-",ｂｉｔ増刊, Vol.21, No.4 (1989)

[3] 萩原,丸島,村岡:"並列処理システムにおけるFORTRANプログラムのマクロブロック化の評価",第３５回情処全大,1C-4 (1987)

[4] E.Riseman, and C.Foster:"The Inhibition of Potential Parallelism by Conditional Jumps", IEEE Trans. Comput., Vol. C-21, No.12, pp.1405-1411 (1972)

[5] J.R.McGraw:"The VAL Language : Discription and Analysis", ACM Trans. on Programming Language and Systems, Vol.4, No.1, pp.44-82 (1982)

[6] Arvind, K,P. Gostelow, and W. Ploufe:"An Asynchronous Programming Language and Computing Machine", TR114a, Dept. Computer Science, Univ. California Irvine (1978)

[7] 島田,関口,平木:"データフロー言語ＤＦＣの設計と実現", 信学論(D),Vol.71-D, pp.501-508 (1988)

[8] D.D.Gajski, D.A.Padua, D.J.Kuck, and R.H.Kuhn:"A Second Opinion on Data Flow machines and Languages", Computer, pp.58-69 (1982)

[9] D.J.Kuck, R.H.Kuhn, D.A.Padua, B.Leasure,and M.Wolfe : "Dependence Graphs and Compiler Optimizations", Proc. of the 8th ACM Symp. on Principles of Programming Languages, pp.207-218 (1981)