計算機アーキテクチャ　92－1
（1992．　1．　23）

PNスーパスカラプロセッサ用コンパイラの構築

鈴木知美　有田隆也　曽和将容
名古屋工業大学

PNスーパスカラプロセッサは機能分割されたユニットで構成されており、各ユニットが適切に同期をとることにより細粒度並列処理を可能としている。機械語命令は、ユニット間の同期用にタグ付けされており、実行前に各ユニットに格納される。このように静的にスケジュールするため、抽出する細粒度並列性はコンパイラに大きく左右される。

本稿では作成中のPNプロセッサ用コンパイラの全体構成を示すとともに、PNプロセッサ特有の性質を利用したコード生成手法について述べる。構文木からはデータフローコードという中間コードを生成する。各ユニットへの命令割り振りにはリストスケジューリングを用いた。基本ブロック間の並列性を抽出し先行関係を適切に保証することにより、基本ブロック間のオーバラップ実行を可能としている。

A Compiler for the PN Superscalar Processor

Tomomi SUZUKI, Takaya ARITA, Masahiro SOWA
Nagoya Institute of Technology, Nagoya, Japan

The PN superscalar processor consists of function-partitioned units. Fine-grained parallel execution is realized by the high speed static synchronization mechanism. Instruction streams which contain tags for synchronization are assigned to the function units by the PN compiler. It mainly depends on the compiler for a statically scheduled superscalar processor like the PN processor to draw fine-grained parallelism efficiently from programs.

This paper describes the basic structure of the PN processor and presents the method of code generation taking advantage of the characteristics of the PN processor. The PN compiler generates the intermediate code named dataflow code. List scheduling techniques are used for allocating instructions to each function units. This compiler makes it possible to overlap the execution of instructions belonging to different basic blocks by securing proper data dependence across basic blocks.

1 はじめに

ＰＮプロセッサ[1][2]は機能分割されたユニットによる並列処理と各ユニット間の実行制御方式[3]に大きな特徴がある。ＰＮプロセッサの機械語命令は同期用にタグ付けされており、実行前に各ユニットの命令メモリに格納される。このように静的にコードスケジュールするため、抽出する細粒度並列性はコンパイラに大きく左右される。

命令を各ユニットに割り振るため、構文木からデータフローコードを生成している。各ユニットへの命令割り振りにはリストスケジューリングを用いた。ＰＮプロセッサの同期方式の性質を利用し、基本ブロックをこえた先行関係を適切に保証することにより、基本ブロック間のオーバラップ実行を可能とした。本稿では作成中のＰＮプロセッサ用コンパイラの全体構成を示すとともに、ＰＮプロセッサ特有の性質を利用したコード生成手法について述べる。

2 ＰＮスーパスカラプロセッサの概要

図１にプロトタイプＰＮプロセッサの構成を示す。

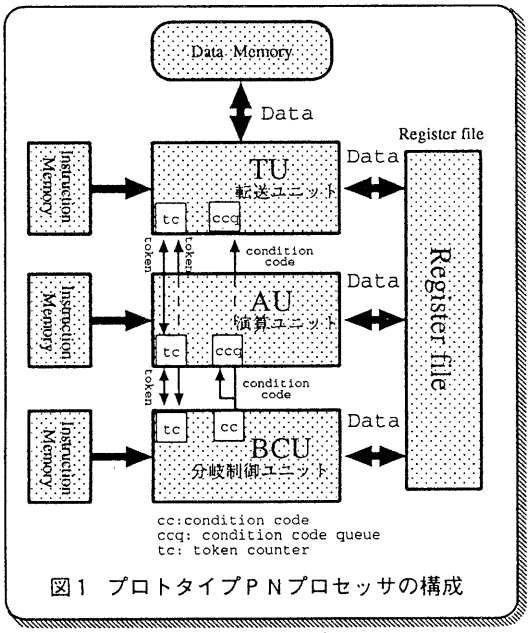

図１ プロトタイプＰＮプロセッサの構成

プロトタイプでは転送ユニット、演算ユニット、分岐制御ユニットと機能が三分割されている。これらのユニットはそれぞれ主に従来のプロセッサにおけるＬＯＡＤ／ＳＴＯＲＥ命令、演算命令、コンディションコード生成命令を実行する。これらの命令は、実行前に（静的に）各処理ユニットの命令メモリに格納される。なお、コンディションコード生成命令により生成されたコンディションコードは各処理ユニットのキューに蓄えられ、条件分岐命令などで使用される。ユニット間のデータ転送は、ユニット間で共有されているレジスタファイルを介して行なわれる。各ユニットは実行順序制御のため実行許可の信号（コントロールトークン）をユニット間で送受する。このため各ユニット間に通信線が張られ、各ユニットにはコントロールトークンを蓄えるためカウンタ（トークンカウンタ）が付けられている。トークンカウンタはコントロールトークンが処理ユニットに送信されてくるとインクリメントされ、命令で受信されるとデクリメントされる。トークンが到着していない場合は、トークン到着まで命令実行が遅らされる。このトークンカウンタによる同期機構は、同期の高速化、機構の単純化を目指すものである。

ＰＮプロセッサが実行するプログラムはＰＮプログラムである。ＰＮグラフはＰＮプログラムを表し、ＰＮプロセッサの動作、各ユニットでの逐次実行とユニット同士の並列動作、ユニット間のコントロールトークンの送受を表す。このＰＮグラフはコントロールフローグラフにユニットに関する実行順序を加え変形したものである。図２にＰＮグラフの例を示す。

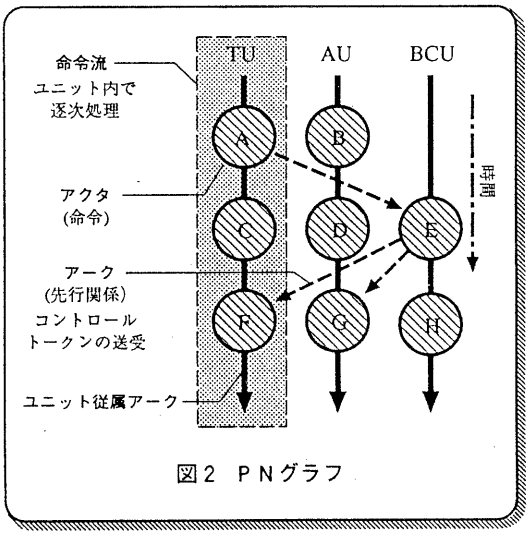

図２ ＰＮグラフ

ＰＮプロセッサでは命令の種類により分割された複数命令流を並列実行する。命令流はユニット

従属アークと呼ばれる縦のアーク上の命令である。命令流は各処理ユニットで逐次処理される。命令流間のアークは実行順序制御のためのコントロールトークン送受を表す。アークのうち命令流間の他のアークによって実行順序が保証されるものは冗長アークと呼ばれる。トークンをカウンタでカウントしているため冗長アークは残しておくことはできない。実行順序制御のためのコントロールトークン送受をＰＮグラフではアークで表すが、機械語命令ではオペコードの前に付けられる前置タグと、後に付けられる後置タグで表す。前置タグは、コントロールトークン受信を表し、後置タグは、コントロールトークン送信を表す。

3　ＰＮプロセッサ用コンパイラの構成

図３に本研究で作成中のＰＮプロセッサ用コンパイラでのコード生成の手順を示す。

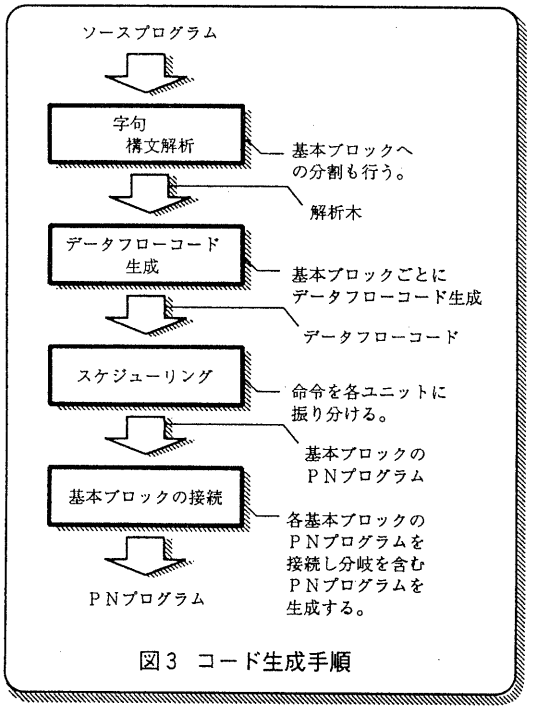

図３　コード生成手順

構文解析により構文木が生成される。ここまでは逐次実行型プロセッサのコンパイラと同様である。ここではプログラムの基本ブロックへの分割も行なう。これは後に使うコード生成法が基本ブロック単位に適用されるからである。また変数の使用状況などの情報も収集している。ＰＮプロセッサ用コンパイラでは、命令の各ユニットへの割り振り、同期用のタグ付けが必要なため構文木からデータフローコードを生成することにした。ここは逐次実行型プロセッサのコード生成部にあたる。データフローコードとは、機械語命令をノードとするコントロールフローグラフである。さらに、ＰＮプロセッサでは、命令を各ユニットに割り振るため、スケジューリングによって、データフローコードをＰＮプログラムに変換する。こうして生成したＰＮプログラムは、基本ブロック[1]のＰＮプログラムである。最後にこれら基本ブロックのＰＮプログラムを分岐、合流に伴う処理を施し、接続することによりプログラム全体のＰＮプログラムを得る。特にここではＰＮプロセッサの性質を利用して基本ブロックをオーバラップ実行できるような同期用のタグ付けを行なった。

4　ＰＮプログラムの生成

命令の各処理ユニットへの割り振りが必要であるためデータフローコードという中間コードを生成している。この処理は基本ブロック毎に行なわれる。この部分には、TREEパタンマッチングとDPを利用したアルゴリズム[4]を使用している。このアルゴリズムは機械語命令生成時における効率的かつ、効果的な命令の選択を可能とする。なお機械語命令を生成するためには変数などのデータストレージの割り付けが必要である。変数をレジスタに割り付ける場合とスタックフレーム上に割り付ける場合では、生成される命令が異なるからである。そこでデータフローコードの生成前にデータストレージの割り付けを行う。ここでは、テンポラリレジスタ[2]の割り付けはしない。テンポラリに割り当てたテンポラリレジスタによる先行関係はスケジューリングや、基本ブロック同士の接続に悪影響を及ぼすからである。データフローコードの例を図４に示す。前述のように、ノードを機械語命令とするコントロールフローグラフである。先行関係によりノード間にアークが付く。命令のオペランドには未確定のテンポラリを残している。

さらに、スケジューリングによって命令を各ユニットに割り振り、データフローコードをＰＮプログラムに変換する。スケジューリング前にテン

＊１　基本ブロック

分岐、合流を含まないプログラムの最長部分である。

＊２　テンポラリレジスタ

機械語命令間でのテンポラリデータ受渡しに用いるようなライフタイムの短い変数に使うレジスタをテンポラリレジスタを呼ぶ。汎用レジスタの一部を使用する。

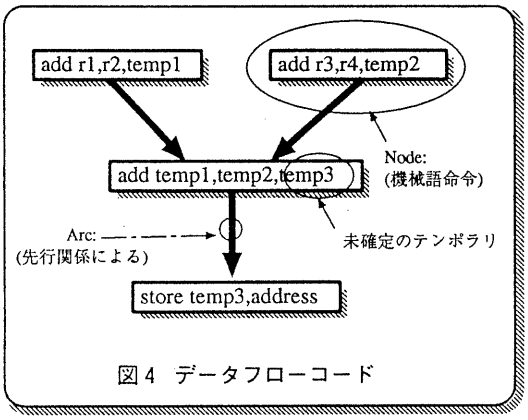

図4 データフローコード

ポラリに使用するレジスタを確定してしまうとスケジューリングは、それらのレジスタによる本来は不要な先行関係の影響を受ける。今回とった方式ではスケジューリングの前にはテンポラリにレジスタが割り当てられていないので、テンポラリによる本来は不要な先行関係の影響を受けない。スケジューリング中にテンポラリレジスタは仮割り付けされる。これは使用するテンポラリ数をテンポラリレジスタとして使用できる汎用レジスタ数内に抑える処理である。この処理にともない、同一のテンポラリレジスタを使用する命令間に先行関係が発生し必要に応じ同期用のタグが付加される。この時点ではまだテンポラリは未確定であり、テンポラリレジスタをテンポラリにどのように対応させることもできる。未確定にしておくのは基本ブロック接続時のためである。スケジューリングは基本的にはリストスケジューリングを使用した。ＰＮプロセッサが非均質のユニットで構成されているという性質を利用したスケジューリング法も考案されている[5]。

スケジューリングによって生成されるのは基本ブロックに分割されたＰＮプログラムである。ＰＮプログラム全体を得るためにそれらを接続する。トークン送受は分岐や合流がいずれの方向でも整合していなくてはならない。整合していないトークンを受け取ることで実行順序が保証されなくなり、誤動作が起こるからである。接続では実行順序を保証しつつトークンを整合させる。

一般に非科学技術計算では分岐命令の出現頻度は高く基本ブロックは小さい。さらにＰＮプロセッサでは命令を複数命令流に分割するので、基本ブロック間の並列性抽出は重要であると思われる。ＰＮプロセッサでは基本ブロックを越えた先行関係はタグ付けで保証可能であるため基本ブロック間のオーバーラップ実行ができる。この特性を利用して、基本ブロックの接続の際には接続する基本ブロック間の先行関係により適切にタグ付けを行ない、基本ブロック間の並列性抽出を行なう。以下に基本ブロック間のオーバーラップ実行を行なうための基本ブロック接続時の処理について述べる。

（1）テンポラリレジスタ割り付け

基本ブロックをオーバラップして実行させる時基本ブロック間には先行関係によるユニットの待ち状態ができることがある（図5）。この基本ブロック間の先行関係が基本ブロック間のオーバラップ実行を妨げる。図5の場合ではＤ－ＥとＤ－Ｉの先行関係によるアークがあるが、Ｄ－Ｉのように待ちになるまでに余裕のあるアークの付け方ではたとえ実行タイミングが スケジューリング時の見つもり[3]からずれても待ちを生じていない。

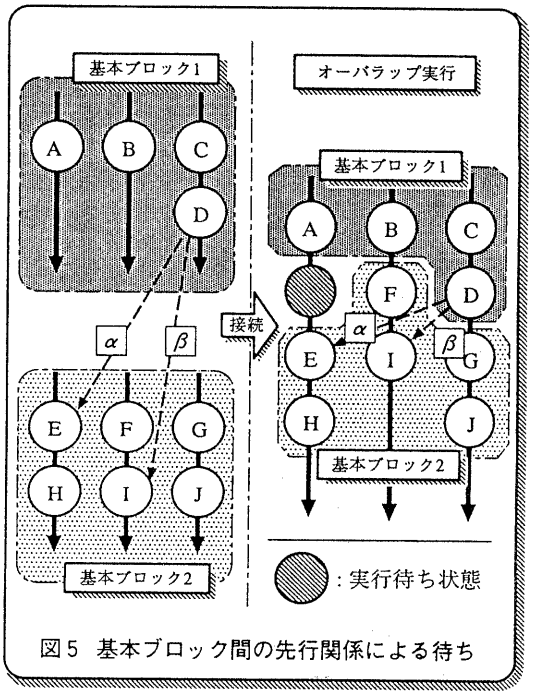

図5 基本ブロック間の先行関係による待ち

そもそも接続する基本ブロック間の先行関係には避けられないものもあるが、テンポラリに使用するレジスタによる先行関係は操作ができる。テ

＊3 スケジューリング時の見つもり

基本ブロックがどのユニットから使用可能になるかは、それ以前の実行に依存し正確に予想することは困難である。

ンポラリはライフタイムが短く、比較的簡単に、置き換えなどが可能だからである。そこで基本ブロックの接続時にテンポラリレジスタを操作し、テンポラリレジスタによる先行関係の距離が長くなるようにする。そのためにＰＮプロセッサ用コンパイラではテンポラリレジスタは各基本ブロックの接続まで確定させず、スケジューリング時にも使用される数だけを合わせている。テンポラリレジスタの割り付けは基本ブロックの接続時にブロック間の先行関係を考慮して決定している。具体的には、接続時に後続ブロックの先頭で使用されているテンポラリレジスタを、先行ブロックの末尾で使用しないようにしている。この時点では基本ブロック内で使用しているテンポラリ数は、テンポラリレジスタ数以下なのでテンポラリとテンポラリレジスタの対応を決めれば良い。この操作により、基本ブロック間のオーバラップ実行がテンポラリに使用されているレジスタによって妨げられないようになる。

この方法には同期を減らす効果もある。ＰＮプロセッサでは他のアークで先行関係が既に保証されたアークを冗長アークと呼び削除するが、待ちになるまで余裕を持ったアークの付け方では間に他のアークを含みやすく冗長アークとなりやすいからである。

（2）タグ付けと命令のコピー

基本ブロックの接続時には複数の基本ブロックとの間で先行関係を保証し、トークン送受を整合させるため、本来は不必要な先行関係を付加しなければならなくなることがある。実行時にアークの付け方が判明するようなアークを動的アークと呼ぶ。このようなアークは分岐時や合流時に発生する（図6（A））。

図6（B）と図6（C）はそれぞれ基本ブロックＡから基本ブロックＢとＣに分岐が起こった場合である。図6（B）と図6（C）では発生する先行関係が異なり、タグ付けを必要とする命令も異なる。合流についても、同様である。ＰＮプロセッサでは静的にコードスケジュールするため、このような場合、先行関係をそのまま保証することはできない。そこでトークン送受を整合させるため新たにアークを付加し、動的アークを冗長なアークにして取り去る、分岐などの方向によらずアークを受け取らせるなどの処理をする。ここで付加したアーク（先行関係）は、付加された分岐、合流の基本ブロック間にとって本来ならば不必要な先行関係である。不必要な先行関係は基本ブロック間の並列性抽出を妨げる。また、合流の場合

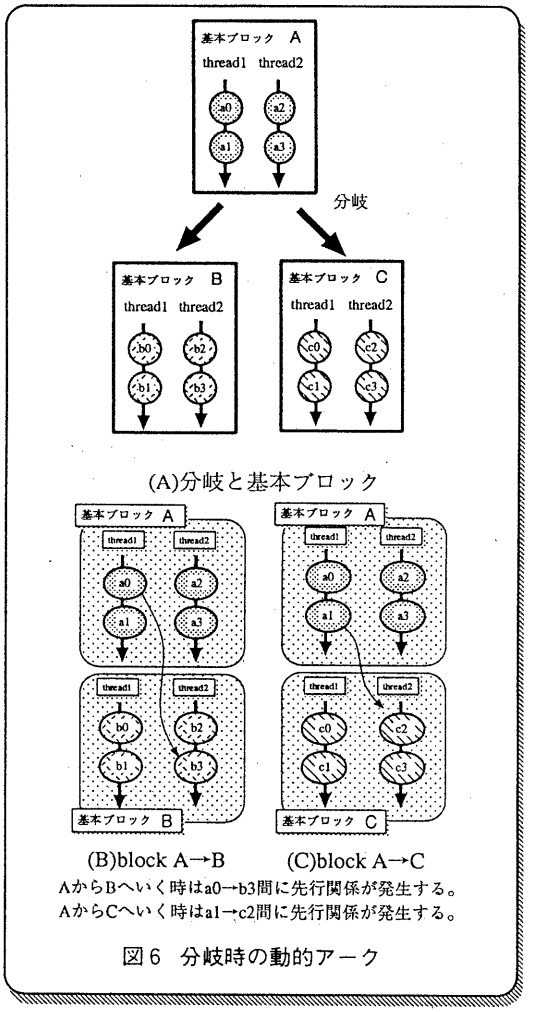

図6　分岐時の動的アーク

は被合流基本ブロックの変化は他の基本ブロック間の接続にも影響を与え接続処理を複雑にする。

そこで被合流基本ブロックのタグの付加される命令までを、合流する基本ブロックの分岐前に複写する方法をとる（図7）。これにより合流に関しては不必要な先行関係を付けずに済む。

図7では、被合流基本ブロック２の命令｛D、E、F、I｝が被複写命令である。これを合流基本ブロック１の分岐命令前に複写し、基本ブロック１からの分岐先を複写した命令のあとに変更している。オリジナルの被合流基本ブロックの変更を必要とせず、各合流、被合流基本ブロック間接続が独立に行なえるため、命令への不必要なタグ付（先行関係付加）をせずに済む。また、接続の処理も単純になる。

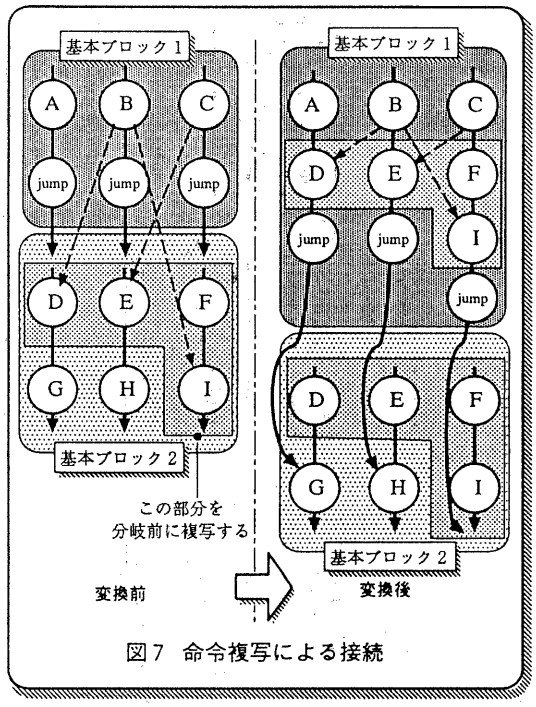

図7 命令複写による接続

5 おわりに

以上作成中のＰＮプロセッサ用コンパイラの構成を示し、ＰＮプロセッサの特徴を利用したコード生成法について述べた。本コンパイラはほぼ完成しており、現在、最終的なデバッグを行なっている。今後、最適化の効果を調べるとともにより効率的なコードを生成するような拡張を検討していく。

参考文献

[1]曽和将容、有田隆也、河村忠明、高木浩光、"機能分割型プロセッサによる複数命令流実行方式"、電子情報通信学会論文誌 D-I, Vol.J-73-D-I, No.3, pp.280-285(1990).

[2]田崎明久、"ＰＮコンピュータのアーキテクチャに関する研究"、昭和６１年度群馬大学工学部情報工学科卒業論文(1986).

[3]T.Arita and M.Sowa, "High Speed Synchronization for a Statically-scheduled Superscalar Processor", International Journal of High Speed Computing,Vol.3,No.1,pp.77-87,World Scientific(1991).

[4]Aho,A.V.,and Ganapathi,M.,and Tjiang,S.W.K., "Code Generation Using Tree Matching and Dynamic Programming", ACM Trans. Program. Lang. Syst., Vol.11 No.4, pp. 491-516(1989).

[5]李 鼎超、有田隆也、曽和将容、"機能別小並列コンピュータ用プログラムの一構築法"、電子情報通信学会技術研究報告、CPSY90-84, PP. 9-14 (1991).