計算機アーキテクチャ　92－3
（１９９２．　１．　２３）

命令パイプライン間のステージレベル静的順序制御方式の検討

高木 浩光　　有田 隆也　　曽和 将容
名古屋工業大学 電気情報工学科

超並列計算機システムの要素プロセッサアーキテクチャに求められる条件として、制御ハードウェアの軽量化、命令処理時間の変動に対する強化を挙げ、これらを同時に満たすものとして、パイプラインステージレベルでの単純で高速な同期方式について検討する。

A Simple Mechanism for Stage-level Interpipeline Synchronization

Hiromitsu TAKAGI, Takaya ARITA, Masahiro SOWA
Department of Electrical Engineering and Computer Science,
Nagoya Institute of Technology
Gokiso, Showa-ku, Nagoya 466, Japan
E-mail: takagi@craps.elcom.nitech.ac.jp

Architectural requirements for processing elements of massively-parallel computer systems are discussed. Two property, simplicity of the control circuit and resistiveness to dynamic fluctuation of execution timing, are claimed to be essential. Then a simple mechanism for stage-level interpipeline synchronization is presented to meet these requirements.

1 はじめに

機能ユニット数千～数万台規模の超並列計算機を実現するためには、1チップ上にできるだけ多数の機能ユニットを集積し、かつこれらを極限まで有効利用することが重要である。このとき機能ユニット間の並列処理方式として適当と考えられるのは、ユニット間の通信コストを考慮すると、レジスタファイル共有型の命令レベル並列処理方式である。

命令レベルの並列処理においては、処理プログラムの構造が実行前に確定していなければならないという制約がある。すなわち基本ブロックの大きさが問題の持つ並列性の大きさを決定してしまう。一般に基本ブロックサイズは比較的小規模であろうといわれているが、数値計算などの分野に限れば、比較的大きな基本ブロックサイズを持つ問題も幾つか報告されている[1),2)]。また非数値計算の分野においても、Percolation Scheduling[3)] や Software Pipelining[4)] 等のプログラム変換技法によって、この基本ブロックサイズをある程度大きくすることができると考えられる。いずれにせよ、少なくとも命令レベルの並列性はそのローカリティを活かし、チップ内で高速に処理してしまうべきである。

命令レベル並列処理アーキテクチャとしてはVLIW, Superscalar方式などが一般的であるが、本稿では、いずれの方式にも超並列計算機の要素プロセッサとしては馴染まない点があることを指摘し、これに適したアーキテクチャとして、一般化静的順序制御機構[5)]をはじめとする単純で高速な同期機構を、パイプラインステージレベルの制御に応用することを検討する。

2 各種命令レベル並列処理方式の特徴

2.1 命令レベル並列処理方式の分類

命令レベル並列処理方式の分類法としては、VLIW方式とsuperscalar方式とに大別するといったものが一般的であるが、最近では両者とも様々の形態のものが提案されており、もはや何等かの定性的な性格付けをする用語としては不十分となってきている。ここでは、命令実行順序制御の観点から次の3項目により分類する。

(1) 実行タイミングが確定する時期

a. コンパイル時

同時に実行される命令の組がコンパイル時に確定する方式で、基本的なVLIW、静的ハザード解消型[6)]のスーパスカラ（例えばTORCH[7)]など）がこれに属す。

b. 実行時

命令間の先行制約がハードウェアによって保証され、実行時に各命令の実行タイミングが確定する方式。動的ハザード解消型[6)]のスーパスカラすなわち、インターロック制御、out-of-order実行制御方式などがこれに属す。

(2) 命令間先行関係の検出時期

a. コンパイル時

コンパイル時に得られる命令間先行関係の情報を命令コード中に埋め込む（例えば、VLIW方式におけるno-op命令の配置など）方式。前述の実行タイミングがコンパイル時に決定される方式は必然的にすべてこれに属す。

b. 実行時

オペランドレジスタ番号の比較などにより命令間の先行関係を動的に検出する方式。

(3) 機能ユニット割当及び実行順序割当の確定時期

a. コンパイル時

静的コードスケジューリング、かつ、in-order実行方式がこれに属す。

b. 実行時

動的コードスケジューリング、あるいは、out-of-order実行方式がこれに属す。

2.2 項目(1)-aの問題点

実行タイミングを静的に確定させる方式の問題点としては、その実現方法としてno-op命令の挿入などを採用した場合、オブジェクトコードサイズが増大するためキャッシュサイズ当たりのヒット率が低下するなどの点が指摘されている。しかしさらにもうひとつ重要な問題点として、命令処理時間の動的変動に対して弱いという点が挙げられる。

一般に、静的コードスケジューリングを前提としたアーキテクチャでは、コンパイル時に各命令の処理時間予測に基づいた最適スケジューリングが行なわれる。実際の命令処理時間がコンパイル時の予測どおりであれば最適に近い実行が実現されるのであ

るが、実行時でなければ処理時間が確定しないような命令が含まれていると、余分な待ちが生ずる可能性がある。

VLIWやTORCHのように、機能ユニット間の実行タイミングが静的に確定しているアーキテクチャにおいては、ある１つの命令の処理時間が予測より長くなってしまった場合、以降の命令間のハザード解消を保証するためにそのほかのすべての機能ユニットの実行をも停止させる必要がある。これに対し、動的ハザード解消方式、すなわち実行タイミングを動的に確定させる方式では、少なくとも全実行ユニットが待合せを行なう必要はない。

実行タイミングが静的に確定する方式のこの問題点は、要素プロセッサ間通信命令などの大幅な処理時間変動のある命令を持つ、超並列計算機システムにおいては重大な問題となる（レイテンシに対して弱い）。

2.3 項目(2)-bの問題点

命令間の先行関係を実行時に検出する方式には一般に次のような問題点がある。機能ユニット数が多くなると（すなわち同時に発行できる命令数が増加すると）、比較すべきレジスタ番号のオペランド対がn^2のオーダで増加し、これに相当する数の比較器とバスが必要となるため、ハードウェア量の増大を招く。この問題は特に、要素プロセッサ当たりのハードウェア量を最小限に抑えたいという超並列計算機システムにおいては致命的な問題となる。

もう一つの重要な問題点は、レジスタ番号比較のみによる実行時先行関係検出法には、ハードウェア量の問題を無視しても次のような限界がある。通常、レジスタ番号比較による先行関係検出は、ある限られた範囲（ウィンドウ内）の命令間のオペランド比較によって行なわれる。これは、連続する２つの命令ブロック（１度にフェッチされる命令集合）の間で行なわれるのが一般的であるが、この方式では命令ブロック単位でin-order実行とならざるを得ない。命令処理時間に変動がない場合においてはこれで十分に最適な実行が期待できるのであるが、レイテンシに対して強いことが望まれる超並列計算機システムの要素プロセッサとしては不十分である。また、たとえウィンドウサイズを無限大にしたとしても、レジスタ再利用によって生ずる世代を越えた先行関係検出ができないという限界がある。この限界は、変数名の付け換えにより反フロー従属を排除することで解消できるが、これには多数のレジスタを必要とする。超並列計算機の要素プロセッサとしてはハードウェア量を抑えたいという点、及びマルチポートレジスタのハードウェアコストは高いという点から、レジスタ数自体は少数にならざるを得ないことを考慮するとこの技法は非現実的である。

2.4 項目(3)-bの問題点

データフローマシンに代表される動的スケジューリング・マシンは、命令処理時間の変動に対して最も強いアーキテクチャであると考えられる。命令レベル並列処理においては、ThorntonのScoreboard[8]やTomasuloのReservation Station[9]及びその拡張型[10),11)]などのout-of-order実行メカニズムが、局所データフロー実行を実現するものとしてこれに相当すると考えられる。しかし、これらのハードウェアは、機能ユニット数が増えると、非常に複雑かつ大規模なものとなるため、クロックレートを遅くする可能性があるうえに、要素プロセッサ当たりのハードウェア量を最小限に抑えたい超並列計算機システムには適していない。

3 提案する命令レベル並列処理方式

前章の分類項目(1)(2)に関して、従来の命令レベル並列処理方式のほとんどは、(1)-a,(2)-a もしくは (1)-b,(2)-b という組み合わせのどちらかをとっている。本稿では両者の問題点を同時に解決するものとして、(1)-b,(2)-aという組み合わせの方式、すなわち、命令間先行関係の検出はコンパイル時に行ない、実行タイミングは実行時に確定させるアーキテクチャについて検討する。

また項目(3)の、命令の機能ユニット割当及び実行順序の確定時期については、以下のように決定する方式を考える。

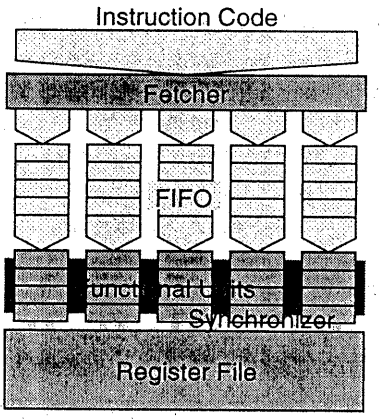
- 各命令が実行される機能ユニットの割当がコンパイル時に決定される。
- 同一機能ユニットに割り当てられた命令間の実行順序がコンパイル時に決定される。

・異なる機能ユニットに割り当てられた先行制約のない命令間の実行順序は実行時に決定される。

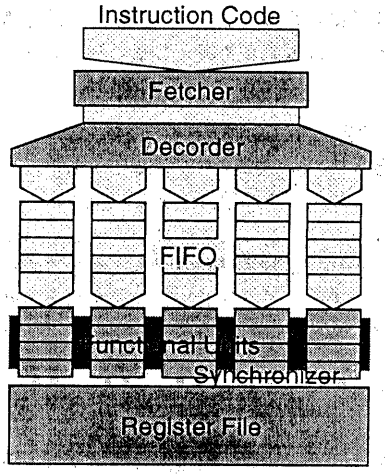
このことは、同一機能ユニット内の命令はin-order実行、異なる機能ユニット間の命令はout-of-order実行であると言い替えることもできる。このような実行形態を仮定すると、静的順序制御方式と呼ばれるクラスの極めて単純な同期機構を利用できることが知られている[5]。

この制御方式は、同一機能ユニットに割り当てられた命令間の実行順序が静的に決められているという制約により、理想的な（ウィンドウサイズによる制限を受けない）局所データフロー実行に比較すると、やや、命令処理時間の変動に対して弱いが、VLIW方式などの静的ハザード解消方式に比べれば十分に優れているおり[12]、また、静的スケジューリング・アルゴリズムの改良によりある程度改善できることも示されている[13]。またこの問題は、有限ウィンドウサイズの局所データフロー実行関しても同様の問題である。

項目(2)-bの動的ハザード解消方式においては、命令間の先行関係はレジスタ番号の比較により実行時に検出するのが一般的であるが、本アーキテクチャでは、コンパイル時のスケジューリング処理の過程で得られている先行関係情報を、単純なグラフ変換アルゴリズムにより圧縮して（後述）オブジェクトコード中に埋め込む方法をとる。これにより、先行関係検出のためのハードウェアが不要となるだけでなく、先行関係検出範囲に限界がないことから、一般的な実行時先行関係検出型のインターロック制御方式に比較しても優れている。



(a)



(b)

図1 基本構成図

4 ステージレベル静的順序制御方式

本アーキテクチャの基本構成を図1に示す。同時にフェッチされるいくつかの命令のそれぞれは、コンパイル時に割り当てられた機能ユニットに対応するFIFOバッファに投入される。このとき、VLIW方式のように、同時にフェッチする命令数を機能ユニット数と同数としフィールドによってどの機能ユニットに割り当てられた命令であるか判定する方式（同図(a)）と、同時にフェッチする命令数を機能ユニット数より少数とし、各フィールド毎にどの機能ユニットに割り当てられたかを示すタグを付加する方式（同図(b)）とが考えられる。後者は前者に比較して、デコード回路が複雑になる、命令コード分配バスのハードウェア量が増えるなどの欠点があり、前者は後者に比較して、命令フィールドが有効に使用されない場合があるなどの欠点がある。本稿ではこの問題について詳しくは触れないが、どちらの方式にせよ、コンパイル時にスケジュールされた順序で命令がFIFOに投入されるものとする。

FIFOに投入された命令は即座に機能ユニットのパ

イプラインに発行される。ここで、オブジェクトコード中に埋め込まれた命令間先行関係情報に基づいて、各パイプライン間のインターロック制御が行なわれる。これには先に述べたように、一般化静的順序制御機構をはじめとする単純で高速な同期機構を用いて実現できる。ここでは同期機構として、一般化静的順序制御機構を採用する場合について説明する。

命令間の先行関係をパイプラインステージレベルで表現すると例えば図2の先行関係グラフで示すことができる。図はパイプライン段数3の場合を示しており、横3列からなる節の一団が一つのパイプラインの実行を表現している。丸で書かれた節は一つのパイプラインステージを表し、縦の並びは同一ステージの連続的実行を表している。白抜き矢印はパイプラインステージ再利用に伴い生ずる「ステージ依存先行制約」であり、この順序でステージが再利用されることを表している。薄灰色の矢印はパイプライン間を受け渡されるデータに関する先行制約であり、これで結ばれたステージが1つの命令であることを意味している。濃灰色の矢印はステージ間のバッファサイズの有限性によって生ずる先行制約であり、ここではバッファサイズを0としているためこのような先行制約が生じている。丸印のすぐ下に位置する長方形で書かれた節は、そのステージが終了できる条件を表しており処理時間は0と考える。例えば、右端のステージすなわち最終段の終了が延期されたとすると、この先行制約により、左側すなわち次命令前段のステージの終了も延期させられる。黒矢印は、レジスタアクセスに伴うフロー従属、反フロー従属、出力従属などの問題が持つ先行制約を表している。

一般化静的順序制御機構を用いてこの黒矢印で書かれた先行制約を保証する場合、レジスタに読み書きをするステージのすべてを一つの命令流とみなしてトークンカウンタ(TC)・ネットワークを形成することで十分である[1]（命令流内の先行関係は命令バッファのFIFOによって保証される）。ただしグラフの特性からいくつかの簡単化を施すことが可能である。例えば、図中、中央のステージがレジスタを読み出すものとし、右端のステージがレジスタに書き込むものと仮定すると、同一パイプライン内における反フロー従属は、パイプラインの構造上常に保証される先行関係であり、これらの間の関係を保証するTCが不要であることがわかる。また、異なるパイプライン間のレジスタを読み出すステージの間には先行制約はあり得ないことから、これらの間にTCは不要であることがいえる。また、さらにハードウェアコストを削減したい場合には、フロー従属の先行制約を保証するTCのみを設けるものとし、反フロー従属や出力従属については、それが保証されるようなフ

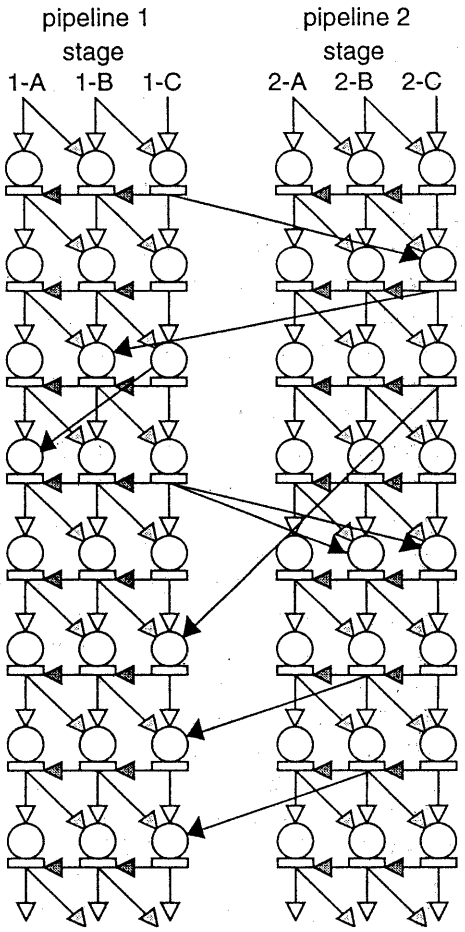


図2 パイプラインステージの
先行関係グラフ表現

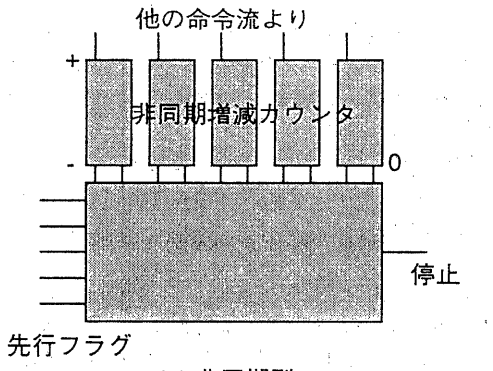
[1]一般化静的順序制御機構の具体的な実現方法については文献5)を参照されたい。

ロー従属の先行制約を付け加えることにより保証できる。ただしこの場合、この先行制約の付加によって、実行タイミングがコンパイル時の予測から変動する環境下において、わずかながら不要な待ちが生ずる可能性が生ずる。

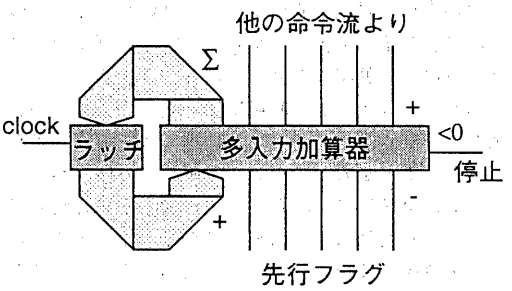
一般化静的順序制御機構を動作させるためにはステージ間に冗長な先行制約が存在してはならない[5]。この冗長な先行制約を削除する処理が、先行関係情報の圧縮処理であるが、この処理はこのようなグラフ表現を用いれば、Transitive Reduction 問題[14]に帰着されるため、容易に実現できる。

文献[5]における一般化静的順序制御機構では、各実行ユニットが非同期に動作することを想定していたが、本稿におけるアーキテクチャでは、各パイプラインは同期式に動作するため、制御機構をさらに単純化することができる。非同期式では、各命令流の実行停止判定回路は図3(a)に示すように、命令流数分の非同期増減カウンタと論理ゲートにより構成されていたが、同期式の場合、同図(b)に示すように、1ビットn入力+mビットの多入力加算器（n:命令流数, m:4～8ビット程度）とmビットレジスタにより構成でき、極めて小規模なハードウェアとすることができる。また、非同期式の場合、先行する処理の実行が完了した後カウントアップパルスを送出し、カウンタなどの遅延を経てようやく後続処理の実行を開始できるため、1クロック程度の遅れが生ずることが避けられなかったが、同期式の場合には、次のクロックでステージが完了すると判断された時点でカウンタ値を計算し、次のステージを停止させるかどうかを決定できるため0クロックでの同期が実現できる。

さらに制御機構のハードウェア量を削減したい場合には、一般化静的順序制御機構の代りに、重複可能なバリア型同期機構[15]を用いることもできる。この同期機構を用いればハードウェア量を極めて小さく抑えることができるが、命令処理時間が変動する環境下で不要な待ちが生ずる可能性が高くなる。この待ちによる全体の実行時間の遅延は命令流数が多いほど大きくなることが示されている[15]。また、重複可能バリア型同期機構を多重化する[16]ことによって両同期機構の中間的性能を中間的ハードウェア量で実現できるので、この性能とハードウェア量とのトレードオフの問題は、システム全体のシミュレーション結果に基づいて必要なだけ同期機構を多重化することで決定すればよい。



(a) 非同期型



(b) 同期型

図3 一般化静的順序制御機構の実現
（パイプライン停止信号生成部）

5 おわりに

超並列計算機システムの要素プロセッサ向き命令レベル並列処理アーキテクチャに求められる条件として、機能ユニット当たりのハードウェア量の軽量化、命令処理時間の変動に対する強化を挙げ、これらを同時に満たすものとして、一般化静的順序制御機構をはじめとする単純で高速な同期機構をパイプラインステージレベルに適用するアーキテクチャを検討した。これにより、部分的なout-of-orderともいうべき実行形態を、Scoreboardや Reservation Stationなどといった複雑なハードウェアを必要とせず実現できることを示した。

本稿では基本ブロック境界における分岐処理手法については触れなかった。ここには制御依存による様々のオーバヘッドが存在すると考えられるが、我々はVLIW計算機における遅延実行方式[17]との融合によりこの問題を解決できると考えている。この点に関しては改めて報告する予定である。

今後の課題としては、例外処理に対する対処法の検討、及び、パイプライン構造と同期機構の構造とが与えられたとき十分に最適に近いスケジュールが実用的な時間内で得られる、ヒューリスティックなスタティック・スケジューリング・アルゴリズムの開発などが挙げられる。

参考文献

1) 笠原博徳, 藤井稔久, 本多弘樹, 成田誠之助: "スタティック・マルチプロセッサ・スケジューリング・アルゴリズムを用いた常微分方程式求解の並列処理", 情報処理学会論文誌, Vol.28, No.10, pp. 1060-1070 (1987).

2) 本多弘樹, 水野 聡, 笠原博徳, 成田誠之助: "OSCAR上でのFortranプログラム基本ブロックの並列処理手法", 電子情報通信学会論文誌, J73-D-I, 9, pp.756-766 (1990).

3) Aiken, A. and Nicolau, A.: "A Development Environment for Horizontal Microcode", IEEE Transactions on Software Engineering, Vol.14, No.5, pp.584-594 (1988).

4) Su, B., Ding, S. and Xia, L.: "GURPR-A Method for Global Software Pipelining", Proceedings of the 20th Annual Microprogramming Workshop, ACM, pp.88-96 (1987).

5) 高木 浩光, 有田 隆也, 曽和 将容: "問題が持つ先行関係のみを保証する高速な静的実行順序制御機構", 情報処理学会論文誌, Vol.32, No.12, pp.1583-1592 (1991).

6) 村上和彰, 久我守弘, 富田眞治: "SIMP（単一命令流／多重命令パイプライン）方式に基づくスーパースカラ・プロセッサの改良方針", 電子情報通信学会技術研究報告, CPSY90-54, pp.97-102 (1990).

7) Smith, M.D., Lam, M.S., and Horowitz, M.A.: "Boosting Beyond Static Scheduling in a Superscalar Processor", Proceddings of 17th Annual International Symposium on Computer Architecture, pp.344-354 (1990).

8) Thornton, J.E.: "Design of a Computer: The Control Dara 6600", Clenview, Illinois: Scott, Foresman, and Co.,(1970).

9) Tomasulo, R.M.: "An Efficient Algorithm for Exploiting Multiple Arithmetic Units", IBM Journal of Research and Development, Vol.11, No.1, pp.25-33 (1967).

10) Weiss, S. and Smith, J.E.: "Instruction Issue Logic in Pipelined Supercomputers", IEEE Transactions on Computers, Vol.C-33, No.11, pp.1013-1022 (1984).

11) 久我守弘, 入江直彦, 弘中哲夫, 村上和彰, 富田眞治: "SIMP（単一命令流／多重命令パイプライン）方式に基づく『新風』プロセッサの低レベル並列処理アルゴリズム", 情報処理学会論文誌, Vol.30, No.12, pp.1603-1611 (1989).

12) 高木 浩光, 有田 隆也, 曽和 将容: "細粒度並列実行を支援する種々の静的順序制御方式の定量的評価", 並列処理シンポジウムJSPP'91論文集, pp. 269-276 (1991).

13) 高木 浩光, 有田 隆也, 曽和 将容: "実行タイミングの動的変動に強い静的プロセッサスケジューリングアルゴリズム", 電子情報通信学会技術研究報告, CPSY90-83 (1990).

14) Aho, A.V., Garey, M.R. and Ullman, J.D.: "The Transitive Reduction of a Directed Graph", SIAM Journal of Computing, Vol.1, No.2, pp.841-848 (1961).

15) 高木浩光, 有田隆也, 曽和将容:"重複可能なバリア型同期のためのスケジューリングアルゴリズムとその性能", 電子情報通信学会技術研究報告, CPSY91-15 (1991).

16) 高木浩光, 山崎憲司, 有田隆也, 曽和将容:"重複可能なバリア型同期の多重化とその効果", 情報処理学会第43回全国大会予稿集, 分冊6, pp.111-112 (1991).

17) 加藤工明, 有田隆也, 曽和将容,: "長命令語(LIW)コンピュータにおける命令実行遅延方式", 電子情報通信学会論文誌, Vol. J74-D-I, No. 9, pp. 613-622 (1991)