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†

高木浩光†† 有田隆也†† 曽和将容††

並列計算機において高い性能を得るためには，高速な命令実行順序制御機構の開発が重要である．本論文では単純なハードウェアによって構成できる，命令のプロセッサ割り当てをコンパイル時に決定する静的順序制御方式について議論する．従来の単純なハードウェアによる静的順序制御機構としてバリア型同期が挙げられる．バリア型同期機構は構成が単純なため高速な制御が可能であるが，すべてのプロセッサが一斉に待ち合わせを行うという同期の性質上，本質的に不要な待ちが生ずるという欠点を持つ．本論文では，静的順序制御方式による実行を並列コントロールフローモデルによって抽象化し，その特性を示すとともに，その特性を利用することによってはじめて可能となる，単純で，かつ，不要な待ちを生じない静的順序制御機構を提案する．提案する制御機構は一般化静的順序制御機構と呼び，プログラムカウンタのほかに，それと同程度に単純なカウンタを任意のプロセッサ間に設け，これらを協調的に動作させることによって実現される．

1. はじめに

並列計算機において効率の良い実行を実現するためには，命令の実行順序制御，例えばプロセッサ間の同期などを高速に処理する必要があり，ハードウェアによる順序制御のサポートが要求される．ハードウェアによる順序制御方式は，命令のプロセッサ割り当てが実行時に行われるかコンパイル時に行われるかによって，動的順序制御方式と静的順序制御方式とに分類することができる．前者の例としては動的なデータフロー計算機など，後者の例としては従来型のマルチプロセッサなどが挙げられる．

一般に静的順序制御方式は，①実行時の命令割り当て処理が必要ない，②単純なハードウェアによって制御機構を実現できる，③先行制御が容易である，などの理由により，動的順序制御方式に比較して高速な制御を期待できるが，動的な先行関係を含む問題を効率よく制御することは困難である．しかし実際の問題には静的な先行関係のみからなる部分が多く存在しており，そのような部分に対しては，静的順序制御方式による高速な制御が効果的であると考えられる．このような観点から本論文では，議論の対象を静的順序制御方式に絞り，実行の対象となる問題は静的な先行関係のみから成ると仮定する．

最も単純なハードウェアで実現できる静的順序制御方式に，共有メモリ型マルチプロセッサで広く用いられているバリア同期[3]がある．バリア同期機構は，すべてのプロセッサの実行が同期点に達したことを検出する極めて単純なハードウェアによって実行することができ，高速な順序制御を実現できる．しかしバリア同期は，すべてのプロセッサが一斉に待ち合わせを行うものであり，個々のデータの依存関係に基づくようなきめ細かい順序制御を行う場合には不要な待ちが発生して効率の悪いものとなる．一方，個々のデータ依存関係に基づいた制御を行う順序制御方式に，データフロー計算機などで用いられているマッチングメモリによる方式があるが，この制御機構では，不要な待ちは生じないが，バリア同期機構に比較してハードウェアが複雑となり高速な制御が期待できない．そこで，バリア同期の構成が単純であるという特長はそのままに不要な待ちを軽減する試みがいくつかなされており，Fuzzy Barrier[4]，一般化されたバリア型同期機構[5),16]などはバリアの概念を拡張することによってこれを実現している（以下これらを総称してバリア型同期と呼ぶ）．ただしこれらの方式も不要な待ちの発生を完全に排除するまでには到っていない[6]．これに対し著者らは，データフローアーキテクチャ単純化のアプローチから，ハードウェアは高速で，かつ，不要な待ちを一切生じない静的順序制御機構の構成法を考察した．

著者らは既に，データフローアーキテクチャ単純化の一手法としてプログラムカウンタの導入[1),2]を提案した．この手法では，問題を並列部分と直列部分とに

† A High Speed Static Synchronization Mechanism without Any Additional Waiting by HIROMITSU TAKAGI, TAKAYA ARITA and MASAHIRO SOWA (Department of Electrical Engineering and Computer Science, Nagoya Institute of Technology).

†† 名古屋工業大学電気情報工学科

分割し，直列部分については，その実行の逐次性を利用しプログラムカウンタによる単純な制御方式を用いることによって，制御の高速化を図ることができた．しかし残された並列部分についても，静的順序制御方式による実行を前提とした場合には，その前提によって生ずる命令の実行順序関係に関する性質を利用して，制御機構を単純化できる可能性がある．

本論文では，静的順序制御方式による実行を並列コントロールフローモデル[7),8)]によって抽象化し，その特性を示すとともに，その特性を利用することによってはじめて可能となる，極めて単純で，かつ，不要な待ちを生じない静的順序制御機構を提案する．提案する制御機構は一般化静的順序制御機構[14)]と呼び，プログラムカウンタとそれと同程度に単純なトークンカウンタによって構成される．この構成はバリア型同期機構に匹敵するほどに単純なものであり，高速な順序制御が可能なものである．

以下，第2章では並列コントロールフローモデルとその性質について述べ，プログラムグラフの変形法などについて説明する．第3章では静的順序制御方式による実行の特性をプログラムグラフの変形という形で表現し，静的順序制御方式として本質的な不要な待ちが生ずるのが如何なる場合であるかについて示す．そして第4章で，不要な待ちが生じないことが保証される範囲内で単純化された静的順序制御機構である，一般化静的順序制御機構の構成法を示す．また第5章では，従来の高速な静的順序制御機構であるバリア型同期機構について，これらの順序制御特性をプログラムグラフの変形という形で表現することにより，それらが不要な待ちを生ずる可能性を持つものであることを示す．最後に第6章では，本制御機構実装時の諸問題について考察し，ハードウェア量の観点から，プロセッサ数30程度の中規模システムであれば，現状の技術で十分実現可能なものであることなどについて述べる．

2. 並列コントロールフローモデル

2.1 特　　徴

問題のもつ並列性を自然な形で表現する手段としてデータフローモデルによるプログラムの記述がある．データフローモデルでは，データの生産消費の関係に基づいてプログラムが記述されるため，問題が本質的に必要とする先行関係のみを表現することが容易である．

データフロープログラムにおけるグラフの有向辺は，命令間の先行関係を表すと同時に命令間のデータの流れをも示しているが，有向辺からデータ流に関する情報を取り除き，それを節内の処理内容記述に含ませると，有向辺と節が示すグラフは，命令間の先行関係のみを表現するものとなる．このようなプログラムを並列コントロールフロープログラムと呼んでいる．先行関係のみしか表していないグラフは，ある条件のもとで自由に変形することが可能となる．このような特徴から，並列コントロールフローモデルを用いることによって，実行順序の制御方式をデータ流の管理方式と分離して独立に設計することを容易にする．以下，実行の対象となる問題は並列コントロールフロープログラムによって与えられるものとする．

2.2 性　　質

並列コントロールフロープログラムから先行関係の情報のみを取り出し抽象化してグラフ化したものを，先行関係グラフと呼ぶ．以下，静的な先行関係のみから成る静的先行関係グラフについて，その定義と性質を簡単に述べる．

（1） 定　　義

先行関係グラフは非巡回有向グラフによって表される．グラフの節は命令を表し，節間を結ぶ有向辺は命令間の先行制約を表している．有向辺が表す先行制約は，「終点側の節の実行は始点側の節の実行が終了しなければ開始できない」ことを意味する．命令の実行終了時には，その命令の実行の影響が，依存関係のある他の命令に及んでいるものとする．

（2） グラフの同値性

ある2つの節間を結ぶ有向辺について，それ自身以外にその2つの節間を結ぶ経路を持つ場合，その有向辺すなわち先行制約は冗長であり（図1参照），それを取り除いたグラフは取り除く前のグラフと先行関係について同値である．

（3） 最小同値先行関係グラフ

すべての冗長な有向辺を取り除いたグラフを最小同

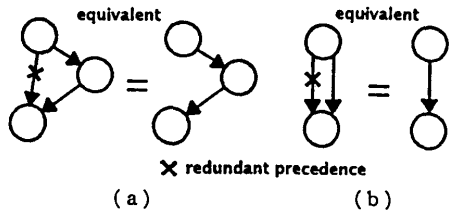


図1 冗長な先行制約とグラフの同値性
Fig. 1 Redundant precedence and equivalence of graphs.

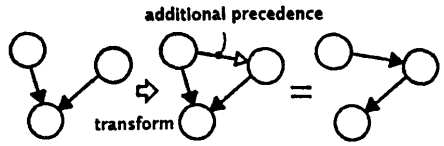


図 2 元のグラフと同じ先行関係を保証する別のグラフへの変形
Fig. 2 Transformation into anothor graph that includes original precedence relations.

値先行関係グラフと呼ぶ．あらゆる先行関係グラフに対して最小同値先行関係グラフは一意に決定される[9]．

(4) グラフの変形

ある2つの節に対し，新たにそれらを結ぶ有向辺を付加する場合，それによって閉路が形成されないならば，付加前の先行関係は付加後の先行関係によっても保証される．このように，閉路を形成しないことを条件に先行制約を付加することによって，先行関係グラフを変形することができる（図2参照）．

(5) グラフの並列性

互いを結ぶ経路を持たない複数の節は同時に実行可能であることを意味する．またこれらを結ぶ有向辺を付加することは，これらの節に対し逐次処理を強いることを意味する．したがって一般に，グラフの変形操作，すなわち先行制約を付加することは，並列性を低くする操作であるといえる（ただし同値なグラフへの変形操作は並列性を変えない）．

3. 静的順序制御方式による実行の特性

3.1 付加的な先行制約の発生要因

データフローモデルなどに基づいて並列に表現されたプログラムも，現実の計算機で実行する際には，記憶セルやプロセッサが有限であることによって並列性が制限される．有限である記憶セルやプロセッサなどの資源は，複数の命令によって共有される必要がある．一般に同一資源に対して複数の命令が同時に占有することはできない．したがって，そこには必ず先行制約が生じ，資源は再利用という形で共有される．この再利用によって生ずる先行制約は，問題が本質的に持つ先行制約とは異なる付加的な先行制約である．このような再利用に伴う先行制約は，その先行制約が生ずる要因によって例えば次のように呼び分けることができる．

(1) 記憶セル再利用に伴う 記憶セル依存先行制約

(2) プロセッサ再利用に伴う プロセッサ依存先行制約

(1)は一般に反フロー従属などと呼ばれる先行制約に相当するもので，(2)は同じプロセッサに割り当てられた命令の実行を逐次化する先行制約である．本論文では，データ流の管理方法については議論の対象外としているので，記憶セル依存先行制約については詳しく触れない．

3.2 プロセッサ依存先行制約

本論文で対象としている静的順序制御方式は，命令のプロセッサ割り当てを実行前に決定するので，付加的な先行制約による並列性の制限は実行前に行われる．またこの制限はプロセッサ資源の有限性によるものであるので，この付加的な先行制約はプロセッサ依存先行制約である．

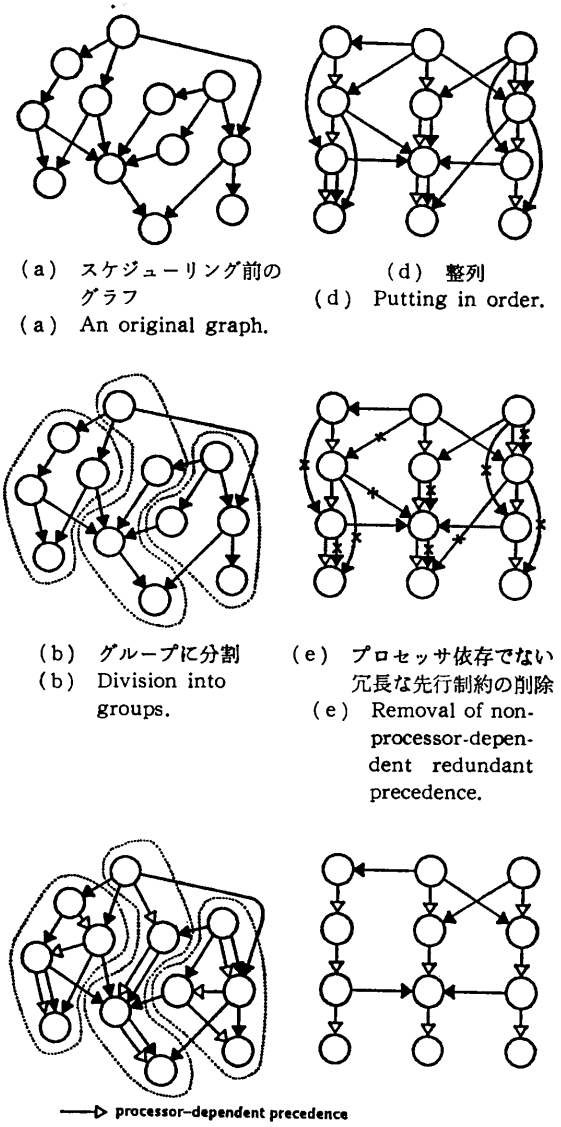
プロセッサ依存先行制約の付加は，コンパイラ等が行う命令スケジューリングの過程で行われるが，この操作は次の2つのステップからなるとみなすことができる．

(1) 先行関係グラフの節をプロセッサ数分のグループに分割する（命令のプロセッサ割り当て）．

(2) グループ内のすべての節を一列化する（命令の順序付け）．

ステップ(1)は同一のプロセッサで共有される命令の組を決定する．ステップ(2)は同一プロセッサで共有される命令の実行順序を決定する．この一列化は，プロセッサ依存先行制約を付加することによるグラフの変形操作によってなされる．例えば，図3(a)のグラフは，プロセッサ数が3の場合，同図(b)のように3つのグループに分割され，図中，白矢印で示した先行制約の付加によって，同図(c)に変形される（同図(d)は節の位置が異なる以外は(c)に同じ）．またこれから，プロセッサ依存でない冗長な先行制約（同図(e)中×印）を取り除くことによって，同図(f)の最小同値先行関係グラフが得られる．この変形には多数の解が存在し，一般には解によってプログラムの実行時間が異なるが，この中から最も実行時間が短くなる解を選択するアルゴリズムについては，最適タスクスケジューリング問題として古くから研究されている[10]．その効率的なアルゴリズムについては，本論文の議論とは独立な問題であり詳しく触れない．また，最小同値先行関係グラフを求めるアルゴリズムについては文献 9) などに述べられている．

静的順序制御方式で順序制御を行うためには，この



(a) スケジューリング前のグラフ
(a) An original graph.
(b) グループに分割
(b) Division into groups.
(c) プロセッサ依存先行制約の付加による一列化
(c) Serialization by appending processor-dependent precedence.
(d) 整列
(d) Putting in order.
(e) プロセッサ依存でない冗長な先行制約の削除
(e) Removal of non-processor-dependent redundant precedence.
(f) スケジューリング後のグラフ
(f) The final graph.

図 3 プロセッサ再利用スケジューリングの実行過程
Fig. 3 A process of static processor scheduling.

ように実行前にプロセッサ再利用のスケジューリングを行うことが必要であり，これは，グラフ上ではプロセッサ依存先行制約を付加することに相当する．付加的な先行制約は並列性を低下させるものであり，場合によって問題が本質的に必要としていない待ちを発生させる可能性のあるものであるが，このプロセッサ依存先行制約の付加は静的順序制御方式を用いる以上避けられないものである．したがって，静的順序制御方式として本質的に不要な待ちが生ずるのは，問題が持つ先行関係に対して，プロセッサ依存でない先行制約が付加される場合のみであるということができる．

4. 一般化静的順序制御機構

4.1 構　　成

本機構が，静的順序制御方式として本質的に不要な待ちを生じないものであるためには，実行前また実行時に，プロセッサ依存以外の先行制約を付加することなく動作するものでなければならない．以下，図3(f)に例示したプロセッサ依存先行制約付加後のグラフを，さらに変形することなく，直接実行の対象とするためにはどのような構造が必要であるかを明らかにすることによって，本機構の構成法を示す．

図3(f)のグラフの特徴のひとつは，一連のプロセッサ依存先行制約によって一列化された，プロセッサ数と同数の命令列が存在していることである．このそれぞれの命令列内の各命令は，同一のプロセッサによって逐次的に実行される．したがって，現在どの命令まで実行が進んでいるかを記憶する構造が各命令列すなわち各プロセッサに存在すればよい．これにノイマン型計算機で用いられているプログラムカウンタ(PC)を用い，各命令列のプロセッサ依存先行制約を保証するための順序制御は，プログラムカウンタの値を増加させる操作によって実現する．

一方，命令列間をまたがる先行関係を保証するための構造に関しては，議論を簡単にするために，以下では任意の2つのプロセッサ間の先行関係のみに着目して考える．

例えば，**図4**の先行関係グラフが与えられた場合，プロセッサ依存先行制約のほかに，「A3の実行はB1の実行が終了してからでなければ開始できない」，「A2の実行はB2の実行が終了してからでなければ開始できない」という先行関係を保証しなければならな

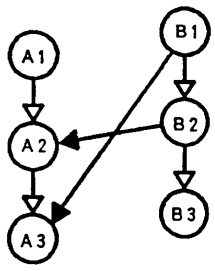


図 4 プログラム例
Fig. 4 An example of programs.

い．したがって，命令列Aを実行するプロセッサAは，A1の実行を終了しA2の実行に移ろうとする際，別の命令列であるプロセッサBのB2の実行が終了するまで，A2の実行の開始を遅らせなければならない．そのためには，B2の実行の終了を示す情報（ここではトークンと呼ぶ）をプロセッサBからプロセッサAに伝送する必要がある．ただしこのとき，B1の実行の終了を示すトークンとB2の終了を示すトークンとが混同されないことを保証しなければならない．その手段としては，命令IDなどを用意してこれを伝送することとし，これを受ける側ではマッチングメモリなどを用いて目的のトークンを探し出す方法が考えられる．しかしこのような方法ではハードウェアが複雑になり，静的順序制御方式の利点である高速な順序制御操作を実現することが難しくなる．

しかし，対象としているグラフには次のような重要な特徴がある．図5(a)の場合，×印の付けられた先行制約は冗長なものであり，取り除いてもグラフの先行関係は同値である．したがって，プロセッサ依存先行制約付加後の最小同値先行関係グラフを求めれば，図5(a)に示すような追い越しをする先行制約や，(b)(c)のような同一命令列に対して複数の先行制約を持つ命令の存在をなくすことができる．この特徴を利用すれば前述のトークン区別の必要性はなくなる．なぜなら，冗長な先行制約のない先行関係グラフが与えられるならば，トークンは対応する命令間で必ず送った順に受理されるため，送られたがまだ現時点では受理していないトークンの数を計数し，これが零でなければ先行する命令の実行は終了していると判定することができる．この方法ではトークンの混同は発生しない．

これを実現するためには，送られたトークン数と受け取ったトークン数の差分を記憶する構造を，命令列間，すなわちプロセッサ間に設け，各プロセッサはその差分が0であるかどうかを検出する構造を用意すればよい．これを通常の増減カウンタを相互に接続することで実現する．このカウンタをトークンカウンタ(TC)と呼ぶ．また，トークンの送信はカウンタに対するカウントアップパルスによって実現し，トークンの受理はカウントダウンパルスによって実現する．図6は任意の2つのプロセッサi, j間のトークンカウンタの接続の様子を示したものである．TC_{ij}はプロセッサiからプロセッサjへ伝送されるトークン数の差分を計数するトークンカウンタであり，カウントアップパルス入力をプロセッサiに，カウントダウンパルス入力および，カウンタ値が零でないことを示す出力をプロセッサjに接続することを示している．

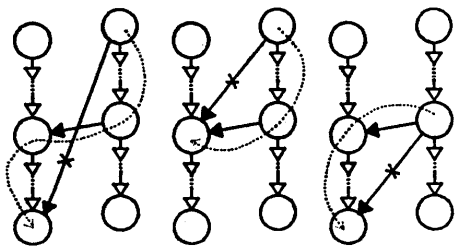


図5 冗長な先行制約を含むグラフのパターン
Fig. 5 Patterns of the graph including redundant precedence.

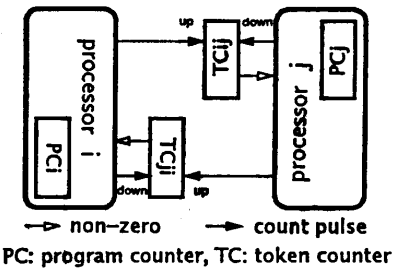


図6 2プロセッサ間の接続
Fig. 6 Connection of 2 processors.

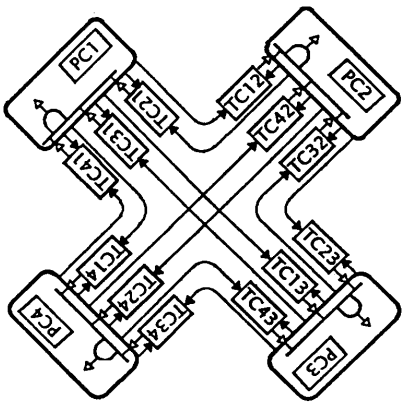


図7 4プロセッサ間の接続
Fig. 7 Connection of 4 processors.

3つ以上のプロセッサに対しては，それぞれの異なる2つのプロセッサ間にこのような構造を図7に示すように並列に設ける．このとき，プロセッサが複数のトークンカウンタに対し同時にカウントパルスを送出できる構造とする．また，複数のトークンカウンタの値がすべて零であることを検出する構造を設ける．

4.2 動　　作

前節で示した構造を用いて次のように順序制御を行

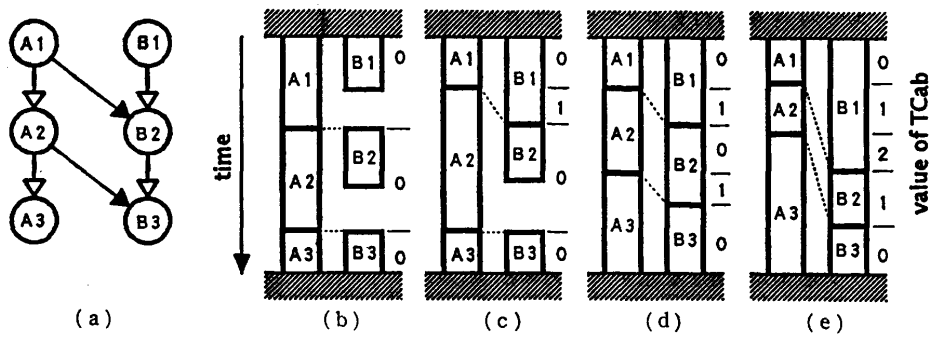


図 8 実行タイミングとトークンカウンタの動作
Fig. 8 An example of execution timing and the value of token counters.

う.

プロセッサ i について

(1) PC_i が指す命令が
プロセッサ j(j≠i) の命令に後続する場合
TC_{ji} の値が1以上になるまで待ち,
その後, TC_{ji} の値を1減ずる

(2) PC_i が指す命令を実行する

(3) PC_i が指す命令が
プロセッサ j(j≠i) の命令に先行する場合
TC_{ij} の値を1増加させる

(4) PC_i を1増加させ(1)に戻る

以上を, i, j について並列に処理する.

図8はトークンカウンタの値の変化の様子を(a)のグラフが与えられた場合を例に示したもので, プロセッサA, B間に接続された TC_{ab} の値と, 命令列A, Bの実行タイミングとの関係を示している. (b)は命令列Bの実行が命令列Aの実行に対して先行している場合である. TC_{ab} の値は通常0であるが, プロセッサAが命令 A1 もしくは A2 の実行を終了した瞬間1になり, プロセッサBの命令 B2, B3 の実行が開始されると同時に0に戻される. (d)(e)の例は命令列Aの方が先行している場合である. 命令 A1, A2 の実行終了と同時に1カウントアップされ, 命令 B2, B3 の実行開始と同時に1カウントダウンされる. この場合 TC_{ab} の値は, プロセッサBが命令 B2, B3 の実行が開始可能であるかどうか TC_{ab} 値を確認する時点では, 常に1以上であり, プロセッサAが実行を中断されることはない. (c)の例はこれらの中間的な場合であり, 実行タイミングによって動作が異なることを示している.

この制御方式は極めてシンプルである. プログラムカウンタによる順序制御の高速性は, 従来のノイマン型プロセッサによって実証済みである. トークンカウンタによる順序制御のオーバヘッドは, 先行する命令(複数の場合を含む)の最も遅い実行終了から後続の命令の実行開始までに要する遅延時間で測ることができるが, これはカウンタの増減に要する遅延時間と数ゲートの論理回路の遅延時間程度のものであると考えられる. この高速性はバリア同期機構などに匹敵するもので, マッチングメモリを用いるような方法に比較して極めて高速な順序制御が可能であると考えられる.

ところで, TC の代わりにセマフォ変数を用いることによって本方式と同様の制御を行うことができることから, 本制御機構は静的な実行順序制御用に用途を限ったサブセットのセマフォをハードウェア化したものとみなすこともできる. この場合, 特定のセマフォの使用法だけをハードウェア化することによって, ①同一のセマフォ (TC) に同時に複数のプロセッサがアクセスすることがなくなり, 排他制御のための調停機構が不要となる, ②特定のセマフォ (TC) にアクセスするプロセッサは固定されているため, アクセス機構が単純化され, 複数のセマフォ (TC) に対する並列アクセスも可能となる, などの一般的なセマフォのハードウェア化では得られない高速化が達成されているといえる.

5. 従来の高速な静的順序制御機構との論理的比較

本機構のように単純で高速なハードウェアで実現できる従来の静的順序制御機構に, バリア型同期がある. 本来バリア同期は fork/join 型の処理を実現するためのものであるが, Fuzzy Barrier[4] や一般化されたバリア型同期機構[5),16)]などは, 個々のデータ依存関係に基づいた制御を行うことによって, プロセッサに生ずる待ちを削減している. しかし, これらのバリア

型同期機構は，本方式と異なり，静的順序制御方式として本質的に不要な待ちを生ずる可能性を持つものである．以下ではこの点を明確にするために，バリア同期，Fuzzy Barrier，一般化されたバリア型同期の3つの静的順序制御機構について，その順序制御特性を先行関係グラフによって表現し，本方式との論理的な比較を行う．

バリア同期のセマンティクスは，「すべてのプロセッサの実行がバリアに達するまで，どのプロセッサの実行もバリアを越えることができない」というものであるが，これを先行関係グラフを用いて形式化すると**図9**のように表現することができる．波線がバリアと呼ばれる同期点を示し，バリア直前の命令のすべてからバリア直後の命令すべてに向かう先行制約（図中黒矢印）によってバリア同期の順序制御特性を示している．

Fuzzy Barrier は，バリア同期の概念を拡張し，バリアに幅があると考えるもので，そのセマンティクスは「すべてのプロセッサの実行がバリア領域に突入するまで，どのプロセッサもバリア領域を抜けて実行を続けることはできない」というものである[4]．その先行関係グラフによる表現は**図 10** のようになる．波線に挟まれ斜線で塗られた命令がバリア領域内の命令を示し，バリア領域直前の命令のすべてからバリア領域直後の命令すべてに向かう先行制約によって Fuzzy Barrier の順序制御特性を示している．

これらは，直接的には図に示したパターンの先行関係しか保証しないという特性を持つものであるといえる．このような特性を持つ順序制御方式を用いることは，問題として与えられた先行関係グラフに，特定のパターンに変形するための，余分な先行制約を付加しなければならないことを意味する．このような順序制御方式に依存した付加的な先行制約をそのほかの先行制約と区別して同期機構依存先行制約と呼ぶことにすると，例えば問題として，**図 11**(a) の先行関係が与えられたとき，これを図中波線で示された位置にバリアを設けて順序制御することは，同図(b)中，斜線で塗られた矢印で示された，同期機構依存先行制約を付加することと等価である．一般に，先行関係の付加によるグラフの変形が，並列性を低下させるものであり，新たな待ちを発生させる要因となることを考慮すると，これらの方式は静的順序制御方式として本質的に不要な待ちを生ずる可能性を持つものであるといえる．

一般化されたバリア型同期機構は，Fuzzy Barrier 方式をさらに拡張したものとみなすことができる[6]．この方式は比較的柔軟な特性を持っており，そのすべての特性を先行関係グラフによって示すことは，誌面の都合上困難であるが，文献 5)，6) に述べられているようにこの方式は，基本はバリア同期に基づいており，各同期には一列化された順序が付けられるという特性を持っている．したがって，例えば問題として

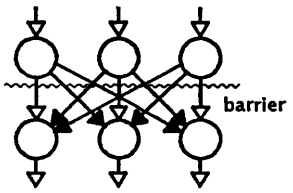
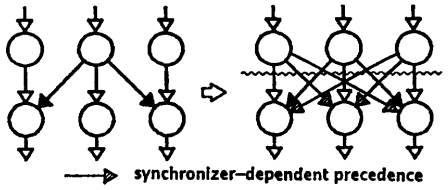


図 9 バリア同期の先行関係グラフ表現
Fig. 9 An expression of barrier synchronization.



(a) (b)

図 11 バリア同期を用いた順序制御のためのグラフ変形例
Fig. 11 An example of transformation for barrier synchronization.

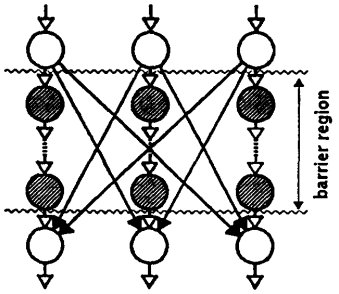
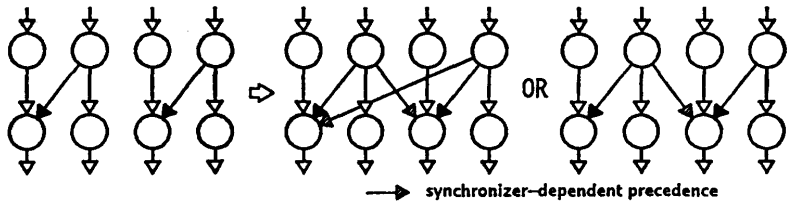


図 10 Fuzzy Barrier の先行関係グラフ表現
Fig. 10 An expression of "Fuzzy Barrier."



(a) (b) (c)

図 12 「一般化されたバリア型同期機構」を用いた順序制御のためのグラフ変形例
Fig. 12 An example of transformation for "generalized barrier-type synchronization."

図 12(a)の2つの独立した先行関係(図中黒矢印)が与えられた場合，これらを同一の同期によって保証するならば同図(b)の同期依存先行制約を付加することと等価であるし，あるいはこれらを異なる同期で保証するものとし，左の先行制約を右の先行制約より先の同期で保証するのであるならば同図(c)の同期依存先行制約を付加することと等価である．このことはバリア同期や Fuzzy Barrier と同様に，本質的に不要な待ちを生ずる可能性があることを意味している（ただし文献6）ではこの問題を解決するために動的順序制御の導入を提案している)．

与えられた問題にこのような同期依存先行制約を付加することによって，バリア同期や，Fuzzy Barrier、一般化されたバリア型同期機構を，本方式上でシミュレートすることができることから，本方式はこれらの順序制御方式を論理的に包含するものであるということができる．

6. 本機構実装時の諸問題に関する考察

6.1 ハードウェア量

本方式のトークン通信用結合網は完全結合である．一般に完全結合の場合，多数のプロセッサ間の接続は困難と考えられる．しかし本方式の通信線の1組は，パルス伝達用の信号線と，カウンタ値が0であることを伝達する信号線とからなるというシンプルなものであり，プロセッサ数 20〜30 程度以下のシステムであれば，物理的な接続はさほど困難ではないと考えられる．

また本方式では，トークンカウンタがプロセッサ数を n としたとき $n(n-1)$ 個必要である．VLSI による実現では，いくつかのプロセッサと，それに直接接続される片一方のトークンカウンタのすべてを，同一チップ上に集積することが適当であるが，この場合，1プロセッサ当たりのトークンカウンタ数は $n-1$ 個である．トークンカウンタのビット数は8〜10ビット程度あれば十分と考えられるので，プロセッサ数が20〜30 程度であれば 150〜300 ビット程度であり，30 個以上のレジスタを持つような最近のプロセッサにみられる集積化技術を用いれば，本機構の集積化による実現に本質的な困難はないと思われる．

6.2 プログラムコードにおける先行関係の表現方法

命令間の先行関係はその情報量を欠落することなくプログラムコード内に埋め込む必要がある．その方法として，タグによる方式と，先行関係表示のための専用命令による方式とが考えられる．タグによる方式は，通常のプログラムメモリに $2(n-1)$ ビットのタグ(nはプロセッサ数，どのプロセッサに先行および後続するかを示す）を付け，命令フェッチと同時にプロセッサ内に取り込む方法である．この方式では，如何なる先行関係が与えられても均一の時間で処理できるという利点があるが，プロセッサ数が多い場合タグビット幅が増大しオーバヘッドとなり得る．専用命令による方式では，情報の圧縮によってこのオーバヘッドを削減できるが，その専用命令の実行自体によるオーバヘッドが生ずる可能性がある．どちらが適切であるかは，システムのプロセッサ数，プロセッサの命令セットアーキテクチャ，プロセッサの命令フェッチ機構などに依存する．

6.3 動的な先行関係への対応

本機構をスーパスカラプロセッサなどの低レベル並列処理[11),12)]に導入する場合には，分岐処理などに伴う動的な先行関係の保証が必要となる．本機構では，送信されたトークン数と受理したトークン数の差分のみに基づいて順序制御が行われるため，制御の分岐点においてジャンプした場合としなかった場合とで差分が異なる場合，制御の合流点以降の順序制御が正しく行われない．この問題点は，制御の分岐点および合流点で，バリア同期と同等の待ち合わせを行うよう先行制約を付加し，トークンカウンタの値を一旦すべて零にすることにより簡単に解決できるが，より効率的な解決法として，付加的な先行関係を持つダミーの命令を挿入することによってカウンタ値を補正する方法などが考えられる．この最適化については様々な方法を検討している[13)]．

6.4 プロセッサのパイプライン制御との整合性

本論文で仮定した実行モデルでは，同一のプロセッサに割り当てられた命令のオーバラップ実行が許されていないが，従来型のプロセッサで用いられているパイプライン制御技術を用いることにより，これらを部分的に並列に実行させることも可能である．ただしその場合，本同期機構が正常に動作するためには，トークン送信（受理）タイミングの順序がオーバラップ実行によって入れ替わることがないことを保証する必要がある．一般的なプロセッサではパイプラインステージの追い越しが起きないことを考慮すると，この条件を満たすためには，少なくとも，トークン送信(受理)をすべての命令において同一ステージで行うものとす

ることで十分である.

7. おわりに

単純なハードウェアによって構成できる静的順序制御方式に着目し，高速で，かつ，本質的に不要な待ちを生じない，一般化静的順序制御機構の構成法を示した．また，従来の高速な静的順序制御方式であるバリア型同期の順序制御特性を，先行関係グラフによって表現することにより，これらが不要な待ちを生ずるものであり，また本方式がこれらを論理的に包含するものであることを示した．

本機構はバリア同期などに比較してハードウェア量は大きくなるものの，制御自体は極めて単純であり高速な制御が可能である．また個々のデータの依存関係に基づくようなきめの細かい順序制御についても不要な待ちを生ずることなく制御できる．

本論文では従来の静的順序制御方式との比較を定性的にのみ行ったが，与えられた問題の実行時間による定量的な評価を，簡単なシミュレーションによって行ったところ，バリア同期方式に対し，1.0〜1.7，平均で1.3倍程度，本方式が優れているという結果を得ている[15]．この定量的な評価についてはより綿密な検討が必要であり，今後の課題である．

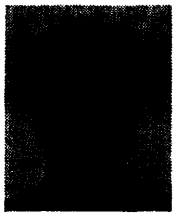
また，本論文では命令実行順序の制御方式についてのみ述べたが，メモリアクセスなど，データ流の管理方式との整合性についての考察なども今後の課題として残される．

謝辞 日頃ご討論いただく曽和研究室の皆様に感謝いたします．

参考文献

1) 曽和将容，羅 四維：データフローコンピュータにおける直列処理の高速化とノイマンコンピュータとの速度比較，情報処理学会論文誌，Vol. 27, No. 5, pp. 533-540 (1986).
2) Sowa, M.: A Method for Speeding up Serial Processing in Dataflow Computer by Means of a Program Counter, *Comput. J.*, Vol. 30, No. 4, pp. 289-294 (1987).
3) Stone, H. S.: *High Performance Computer Architecture*, Addison-Wesley Publishing Company (1987).
4) Gupta, R.: The Fuzzy Barrier: A Mechanism for High Speed Synchronization of Processors, *Proc. Third Int. Conf. ASPLOS*, pp. 54-63 (1989).
5) 松本 尚：細粒度並列実行支援機構，情報処理学会研究報告，89-ARC-77-12, pp. 91-98 (1989).
6) 松本 尚：一般化されたバリア型同期機構の諸問題について，並列処理シンポジウム型同期機構の諸問題について，並列処理シンポジウム JSPP '90 論文集，pp. 49-56 (1990).
7) Sowa, M.: A Control Flow Parallel Computer Architecture, *IPS Japan, SIGARCH*, 48-2 (1983).
8) Trealeven, P. C., Hopkins, R. P. and Rautenbach, P. W.: Combiniing Data Flow and Control Flow Computing, *Comput. J.*, Vol. 25, No. 2, pp. 207-217 (1982).
9) Aho, A. V., Garey, M. R. and Ullman, J. D.: The Transitive Reduction of a Directed Graph, *SIAM J. Comput.*, Vol. 1, No. 2, pp. 131-137 (1972).
10) Hu, T. C.: Parallel Sequencing and Assembly Line Problems, *Oper. Res.*, Vol. 9, pp. 841-848 (1961).
11) 富田眞治：並列計算機構成論，昭晃堂 (1986).
12) 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).
13) 小島聖司，有田隆也，曽和将容：並列計算機におけるカウンタを用いた同期削減手法，第 43 回情報処理学会全国大会論文集，分冊 6，pp. 113-114 (1991).
14) 高木浩光，有田隆也，曽和将容：細粒度並列計算機のための一般化静的順序制御機構，電子情報通信学会技術研究報告，CPSY 89-85 (1990).
15) 高木浩光，河村忠明，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の構成法，並列処理シンポジウム JSPP '90 論文集，pp. 57-64 (1990).
16) 松本 尚：細粒度並列実行支援マルチプロセッサの検討，情報処理学会論文誌，Vol. 31, No. 12, pp. 1840-1851 (1990).

(平成2年 8 月 21 日受付)
(平成3年 10 月 3 日採録)



高木 浩光（学生会員）

1967年生．1989年名古屋工業大学工学部電気情報工学科卒業．1991年同大学大学院工学研究科博士前期課程（電気情報工学専攻）修了．現在同後期課程に在学中．並列処理，計算機アーキテクチャ，スケジューリング・アルゴリズム等に関する研究に従事．電子情報通信学会会員．

有田 隆也（正会員）

1960年生．1983年東京大学工学部計数工学科卒業．1988年同大学大学院工学系博士課程修了．工学博士．同年名古屋工業大学工学部電気情報工学科助手．並列計算機アーキテクチャ，プログラミング方法論に興味を持つ．IEEE，電子情報通信学会各会員．

曽和 将容（正会員）

1974年名古屋大学大学院博士課程（電気電子専攻）修了．同年群馬大学工学部情報工学科助手．1976年助教授．1987年名古屋工業大学教授．この間，並列処理，計算機アーキテクチャ，特にデータフロー計算機，制御フロー計算機など次世代コンピュータの研究に従事．工学博士．IEEE，ACM，電子情報通信学会各会員．