計算機アーキテクチャ　92－2
（１９９２．　１．　２３）

静的順序制御機構を利用した大域的な最適化手法

小島　聖司　　有田　隆也　　曽和　将容
名古屋工業大学　電気情報工学科

細粒度の並列実行を可能とする静的順序制御機構が最近研究されてきている。この種の制御機構は実行前のプログラム解析を前提としているので、条件分岐で接続が異なる基本ブロック間では余分な順序関係をプログラムに付加する必要があった。

本稿ではそのような順序関係を最小限にするプログラム作成手法である「ダミー方式」の提案を行なう。本手法を用いることにより、連続して実行される基本ブロック間の命令が効率的にオーバラップして実行される。提案手法を用いることによりプログラムの実行時間を数％向上させることができることをシミュレーションによって示す。また、その提案手法に適したスケジューリング法についても考察を行ない、それによりさらに４％程度、性能を向上させることができることも示す。

A Synchronization Method using High Speed Static Synchronization Mechanisms

Shoji Kojima　Takaya Arita　Masahiro Sowa
Department of Electrical Engineering and Computer Science, Nagoya Institute of Technology

This paper proposes a new synchronization method using high speed static synchronization mechanisms for fine-grained parallel execution. Execution of instructions in successive blocks can be overlapped and performance is improved,because dynamic dependency between basic blocks are properly resolved in this method. Simulation results show 4-10% speedups by this synchronization method.

1．はじめに

近年並列計算機の研究が盛んに行なわれている。並列計算機において効率の良い実行を行なうためには、命令の実行順序の制御法が重要であり、その機構として多くの静的順序制御機構が提案されている[1)2)3)]。これは実行前にプログラムの解析を行ない、依存関係の抽出とスケジューリング等を行なってしまうものである。必要な処理の多くを実行前に行なっておくこの機構では計算機ハードウェアを単純にすることができ、それにより高速に動作させることが可能である。また、最近の研究では、一般化静的順序制御機構[1)]のような細粒度の並列処理を効率良く実現できる機構も提案されている。

静的順序制御機構では、条件分岐などのプログラムの流れの制御を挟んで実行される命令間の順序関係の保証に、バリア同期手法が利用されることが多かった。しかし、この方法では流れ制御時に実行ユニットに停止状態を生じさせるため、性能の低下を引き起こすことになる。そこで、細粒度並列処理において、このような順序制御を効率的に行なう手法が必要となってきた[4)]。

本論文では、その手法として「ダミー方式」[5)]を提案し、これに適したスケジューリング手法を提案する。「ダミー方式」を利用することにより、バリア同期手法ではなく、同一基本ブロック内の命令間の順序制御と同様な手法でプログラムの流れの制御を挟んで実行される命令間の順序制御を行なえるようになる。その結果、連続して実行される基本ブロック内の命令がオーバラップして実行できるようになり性能を向上させることができる。提案するスケジューリング法により、このオーバラップを効率良く行なえるようになる。

以降、2章では静的順序制御機構の概要について述べ、3章では流れの制御により発生する「動的な順序関係」について述べ、既存の手法で生じるオーバヘッドについてとり上げる。そして、4章で本提案手法である「ダミー方式」とそれに適したスケジューリング手法について述べ、5章でその性能をシミュレーションによって評価する。

2．静的順序制御機構

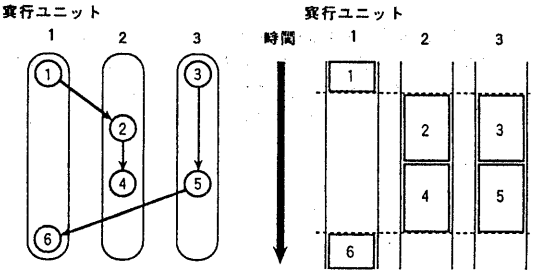
2．1．静的順序制御機構とその問題点

静的順序制御機構では、プログラムを実行する前のコンパイル時に命令間の依存関係の解析を行なう。そして、その結果に基づいて、実行ユニットへの割りつけと実行時に順序制御機構が行なう同期処理内容の決定を実行前に行なう。そのため、実行時にはハードウェアによる依存関係の抽出は必要なく、ハードウェアの簡略化および高速化を実現できる可能性がある。

バリア同期機構は最も簡略化された静的順序制御機構である。この機構では全実行ユニットが待ち合わせを行なう同期により順序制御を行なう。そして、この機構の構成に必要となるハードウェアは、論理的には全実行ユニットからの1ビットの信号線の論理積をとる回路だけで十分である。

しかし、ハードウェアが簡略であるがゆえに、実行時に不必要な実行ユニットの停止状態を引き起こしてしまう場合がある。図1（a）のようなプログラムを、バリア同期機構しか持たない計算機で実行する場合を考える。図で円が命令、矢印が命令間の順序関係を表している。この図ではスケジューリングがすでに行なわれており、細線で囲まれた命令群は同一の実行ユニットに割り付けられていることを示している。例えば、命令1と2の順序関係を保証するために全実行ユニットが待ち合わせを行う。その時の実行タイミング例を図1（b）に示す。図1（b）で、破線を引いたタイミングでバリア同期を行っている。この図から判るように命令1，2間の順序関係を保証するための同期により、これらの命令を実行しない実行ユニット3に停止状態が生じている。

このような不要な停止状態を減らすために種々の静的順序制御機構が提案されている。バリア同期の待ち合わせタイミングに幅を持たせる拡張を行なったファジーバリア[2)]、順序関係に関連した命令を実行しない実行ユニットを待ち合わせから解放することができる一般化バリア[3)]、さらに命令単位での順序制御を可能にした一般化静的順序制御機構[1)]などである。



（a）プログラム　（b）実行タイミング

図1　バリア同期機構によるプログラム実行の様子

2．2　一般化静的順序制御機構

バリア同期機構は、順序関係を保証するために全ての実行ユニットの待ち合わせによる同期を行なう。それに対し、一般化静的順序制御機構は、順序関係の存在する命令を実行するユニットだけが、ある一定の手続きで同期を行なう。

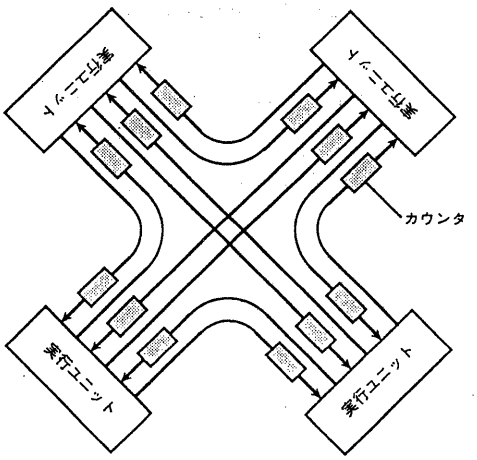


図2　一般化静的順序制御機構

図2は一般化静的順序制御機構の基本構成を示したものである。これは実行ユニット数が4個のものである。各実行ユニットはカウンタを介して完全結合をしている。このカウンタは

順序関係を持つ命令を実行する時に更新される。例えば、ある順序関係を持つ２つの命令が存在する場合を考える。先行して実行される命令を実行する実行ユニットは、その命令を実行した後に順序関係のある命令を実行する実行ユニットとつながっているカウンタを増加させる。すると後続側の命令を実行する実行ユニットは、先行する命令が実行終了したことをカウンタの数により知ることができる。カウンタの値を監視し、更新されたことを確認して後続命令を実行する。

プログラムの実行の際にこの同期処理を行なうため、順序関係の存在する命令に情報が付加されている。この情報は、命令を実行する前にカウンタを参照するためのものと、命令実行後にカウンタを更新するためものとの２種類ある。実行ユニットはこの静的に決定された情報をもとに決められた同期処理を行なうだけでよい。

この順序制御機構では、先行する命令を実行する実行ユニットは後続する命令を実行する実行ユニットの状態を知る必要はなく、割り付けられた命令を連続して実行できる。これにより、この順序制御機構を持つ計算機では、バリア同期機構を持つ計算機と異なり、不要な順序関係を付加することなく命令間の順序関係を保証することができ、順序関係の保証を論理的には最適に実現できる。

図３は、図１（ａ）のプログラムをこの機構によって実行した場合の実行タイミングを示している。破線の引かれたタイミングで実行ユニットが同期を行なっている。バリア同期機構で生じていた、実行ユニット３の停止状態が削減されていることがわかる。

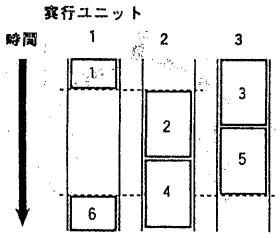


図３　一般化静的順序制御機構による
プログラム実行の様子

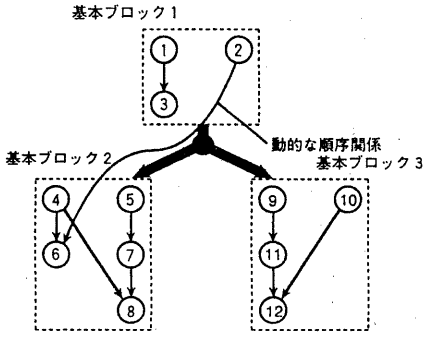


図４　動的な順序関係を含むプログラム

３．動的な順序関係

３．１　動的な順序関係

２つの命令の実行が、分岐などのプログラムの流れの制御を挟んで行われる位置関係に存在する場合、この２命令間に存在する順序関係を「動的な順序関係」と呼ぶことにする。図４は、実行時に決定される条件によりプログラムの流れの制御が行なわれるプログラムである。破線で囲まれた部分が基本ブロックである。基本ブロック間を結ぶ太い矢印がプログラムの流れを示しており、プログラムは基本ブロック１から基本ブロック２もしくは３のどちらか一方に流れる。この図の命令２と命令６の間の順序関係が「動的な順序関係」である。

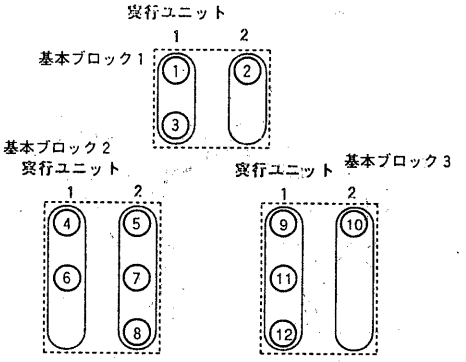


図５　スケジューリングを行なったプログラム

３．２　バリア同期手法を利用した動的な順序関係の保証手法

図５は図４のプログラムを実行ユニット数２でスケジューリングしたものである。図で細線で囲まれた命令群が同じ実行ユニットで実行される。静的順序制御機構で命令２と６の間の順序関係を保証する方法を考える。静的順序制御機構は、ある命令実行に付随して行なう同期処理の内容を実行時に動的に変えることができない。しかし、図のプログラムでは、命令６が実行される条件の場合のみ、動的な順序関係を保証するための同期処理が必要であり、それ以外の条件では必要でない。すなわち、動的に同期処理の内容を変化させる必要があることになる。そのため、静的順序制御機構では、図に示された順序関係だけで必要な順序関係を保証することができない。そこで、このような問題点を持つ動的な順序関係を保証するため、プログラムの変更を行なう。

動的な順序関係の保証は、プログラムの流れの制御の前後で実行される全命令間に順序関係を付加することにより、実現することができる。そこで、プログラムの流れの制御を行なうときにバリア同期を行ない、その前後の基本ブロック間の命令に順序関係を付加する。

図４のプログラムにこれを行なった場合、図６のように変形されたことと等価である。命令２と命令６の間の動的な順序関係は、破線で示された複数の順序関係により保証されている。

バリア同期手法の考え方を利用した、このような動的な順序関係の保証手法を「バリア方式」と呼ぶ。またプログラムの流れの制御を行なうときに常にバリア同期を行なう方式を「基本的なバリア方式」とする。図６のように「基本的なバリア方式」を用いた場合には、多くの順序関係がプログラムに付加される。図６のプログラムを実行した場合の実行タイミングの例を図７に示す。この図は基本ブロック１の次にブロック２にプログラ

ムが流れた場合のもので、２つの基本ブロック内の命令の実行は全くオーバラップされていない。実行ユニット２は命令２と命令５を実行する間に停止状態を生じている。これは命令３と命令５の間に付加された順序関係によるものであるが、変形前のプログラム自体にこの順序関係は存在しない。すなわち、この停止状態は不要なものである。

「バリア方式」において、この停止状態を削減する「拡張されたバリア方式」を考えることができる。しかし、「拡張されたバリア方式」を用いた場合においてもこの問題を完全に解決することは困難である。

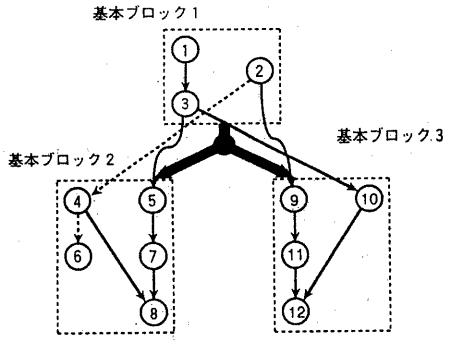


図６　既存手法により変更されたプログラム

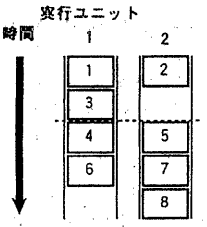


図７　既存手法を用いた場合のプログラム実行の様子

４．動的な順序関係の効率的な保証手法（ダミー方式）

４．１　ダミー方式の概念

動的な順序関係を保証するため、バリア方式では非常に多くの順序関係をプログラムに付加する。しかし、プログラムの流れ制御の時に同期を行なうので、実行ユニットに停止状態が生じ、実行性能が低下することになる。そのため、プログラムへの順序関係の付加を最小限に抑える手法が必要となる[3)]。

図４において動的な順序関係が問題となるのは、命令２と順序関係の存在する命令６が実行されるか否かが、命令２を実行する時には判らないからである。このため、このままでは実行ユニット２は命令２の実行終了後に実行ユニット１との同期処理を行なうべきか否かが動的に変化する。すなわち、命令２と順序関係のある命令が実行ユニット１で必ず実行される場合には上のような問題は生じないことになる。

基本ブロック２にプログラムの実行が流れる場合、命令６が実行される。したがって、実行ユニット２は命令６との順序関係を保証するために実行ユニット１との同期処理を行なう。しかし、基本ブロック３にプログラムが流れる場合には、実行ユニット１で実行される命令群内に命令２と順序関係を保証しなければならない命令は存在しない。そこで、命令２との順序関係を、基本ブロック３の中の実行ユニット１に割り付けられた命令に付加する。例えば、図４ではそのような命令として命令１１を利用することができる。すると、実行ユニット２は、基本ブロック３にプログラムが流れる場合にも、実行ユニット１との同期処理を行なうことになる。すなわち、動的に決定される条件にかかわらず、実行ユニット１で実行される命令との順序関係を保証するために同期処理を行なえることになる。

この方式では、図４のプログラムは図８のように変形される。これにより動的な順序関係による問題点は解決され、静的順序制御機構で順序関係を保証できるようになる。バリア方式は、動的な順序関係を複数の順序関係により保証していた。しかしこの方式の場合、動的な順序関係はそのまま実現され、それが必要となる場合には不要な停止状態が生じることはない。また、付加した順序関係による停止状態もバリア方式に比べて軽減することができる。この方式を、プログラムに最小限のダミーの順序関係を付加することにより動的な順序関係を保証する方式として「ダミー方式」と呼ぶ。

ダミー方式を用いた場合の実行タイミングは図９のようになる。２つの基本ブロック内の命令実行がオーバラップされ、不要な停止状態が削減されている。

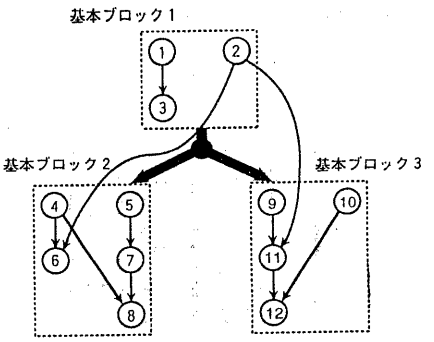


図８　提案手法により変形されたプログラム

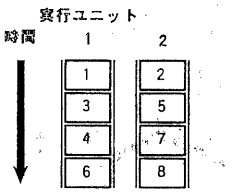


図９　提案手法を用いた場合のプログラム実行の様子

４．２　ダミー方式の実現方法

ダミー方式を用いる場合、保証する動的な順序関係を持つ命令が、どの様な位置関係にあるかを解析しなければならない。図４のプログラムでは、動的な順序関係の存在する２命令は条件分岐を挟んで実行される関係にある。そして、先行して実行されなければならない命令２は実行ユニット２で、後続して実

行される命令６は実行ユニット１で実行される。

基本ブロック３にプログラムが流れる場合、動的な順序関係が必要とならないのは命令６が実行されないからである。そこで、つじつまを合わせるために、動的な順序関係を必要としない場合に制御が流れる基本ブロック３の内部に、命令２と順序関係を持つ命令を作る。この場合、命令６を実行するはずであった実行ユニットと同じユニットで実行される命令を選択する。また、順序制御機構を正しく動作させるためには、付加することになる順序関係が冗長とならないよう注意する。すなわち、命令１２を選択しない。また、付加する順序関係のための同期の自由度を高めるために、なるべく遅い順番で実行される命令を選択する。これにより、図４では命令１１が選択される。

１つの動的な順序関係に対して複数のダミーの順序関係が必要となる場合が存在する。それは、動的な順序関係が複数のプログラムの流れの制御を越えて存在する場合、その制御に関係する全ての基本ブロックにダミーの順序関係を付加しなければならないからである。例えば図１０のような構造を持ったプログラムを考える。図で四角は基本ブロックを表しており、複数の命令が含まれている。図で細線のような位置関係の命令間に動的な順序関係が存在したとする。この場合には２つのプログラムの流れの制御を挟んだ動的な順序関係である。その場合には基本ブロック４と５の両方にダミーの順序関係を付加する命令を用意しなければならない。

動的な順序関係は、条件分岐のような選択によってプログラムの流れが選択される部分だけではなく、分岐後の合流点のような部分にも存在する。その場合には逆に先行側で順序関係を付加するための命令を選択することになる。以上のような処理をプログラム中の全ての動的な順序関係に対して行なう。

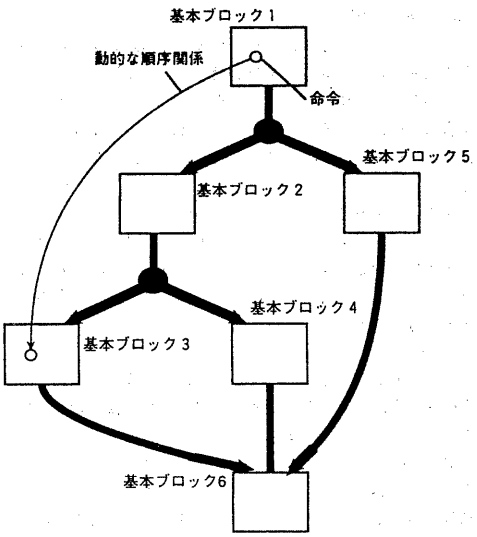


図１０　複雑なプログラム構造例

４．３　ダミー命令の挿入

ダミー方式を用いて動的な順序関係を保証した場合、バリア方式に比べて不要な実行ユニットの停止状態を減らせることを示した。しかし、最適な実行を可能にできない場合も存在する。例えば、図４のプログラムではダミーの順序関係を付加することのできる命令１１が存在したが、このようなダミーの順序関係を付加することのできる命令が存在しない場合も考えられる。その場合にはダミーの順序関係を付加するためにプログラム中にダミーの命令を挿入することになる。このダミーの命令挿入により、その命令の後に実行される命令の実行開始が遅らされる。このような遅れが多く生じる場合にはダミー方式を用いてもあまり性能が向上しない場合も考えられる。

ダミーの命令の実行時間が０とできる場合には、ダミーの命令による問題は生じず、ダミー方式によって変形されたプログラムは、既存の方式であるバリア方式より必ず性能的に優れている。そうでなく最悪の場合には、ダミーの命令によりダミー方式の性能がバリア方式より劣る場合も存在する可能性がある。しかし、スケジューリング時に、ダミーの命令を必要とする場合を減らすように考慮することも可能であり、このようなダミーの命令数は少くできる考えられる。４．２より、他の実行ユニットで実行される命令と順序関係が無い場合に、ダミー用の命令に利用できることが分かる。そこで、スケジューリング時に、その命令と順序関係の存在する命令を実行する実行ユニットを優先して割り付けるようにする。これによりダミーの順序関係の付加に利用できる命令が増えることになるので、相対的にダミーの命令の影響を減らすことができる。

４．４　ダミー方式に適したスケジューリング法

バリア方式により動的な順序関係を保証する場合には、連続して実行される基本ブロック内の命令の実行がオーバラップされないため、基本ブロック内の順序関係だけを用いたスケジューリングで十分であると考えられる。その場合にはＣＰ法などの発見的近似最適なスケジューリング法がそのまま利用できる。しかし、ダミー方式を用いることにより、オーバラップ化されるため、このオーバラップ化に適したスケジューリングの拡張を行なう。これはある基本ブロックのスケジューリングを行なう場合に、そのブロックの直前に実行される基本ブロック内の命令の実行タイミングの予測値を利用するものである。

まず、スケジューリングの前提について述べる。第一に計算機を構成する各実行ユニットは均質の機能を備えていることとする。また命令の実行時間は予測可能であり、計算機の実行時の状態によって変動しないとする。また条件分岐の場合、条件の決定が確率的に予測できるとしている。

条件分岐などの条件により、プログラムの流れは動的に変化するため、プログラム内の命令の実行について実行前に完全に把握することはできない。そのためスケジューリングは基本ブロック毎に行なうものとする。そして、基本ブロック内のスケジューリングに、その基本ブロックを実行する前に実行されると予測される基本ブロックの情報を利用する。

図１１のような構造を持つプログラムのスケジューリングを考える。これは一つのループからなるプログラムである。四角が基本ブロック、矢印がプログラムの流れを表している。基本ブロック２の次に条件分岐により基本ブロック３もしくは４に流れる。また条件分岐の矢印に付加された数字は流れが選択される確率である。この場合は８０％の確率で基本ブロック３、残り２０％の確率で基本ブロック４に流れることを表している。

基本ブロック間の命令の実行を効率的にオーバラップさせるために、ある基本ブロック内の命令のスケジューリングを行なう場合に、先に実行される基本ブロック内の命令がどのようなタイミングで実行終了するかという情報を利用する。例えば図

１１では、基本ブロック２内のスケジューリングするために基本ブロック１もしくは基本ブロック３内の命令の実行終了タイミング情報を用いる。そして、各実行ユニットに割り付けられた命令の実行終了時間から、実行ユニットの利用可能となる時間を予測する。ＣＰ法では、スケジューリングを行なう場合に全実行ユニットが同じ時間から利用可能であるとしてスケジューリングを行なっている。しかし、ここでは先に行なわれると予測される基本ブロックのスケジューリング結果から得た時間を用いてスケジューリングを行なう。これにより、連続して実行される基本ブロック内の命令実行が効率的にオーバラップされる。

図１１では基本ブロック２の前に実行されると予測される基本ブロックは２つ存在する。この場合には、なるべく多く実行される側の基本ブロックのスケジューリング結果を用いる。なぜなら多くの場合にはそちら側からプログラムの流れが来るため、そちら側から流れて場合に効率良く実行できるようにするべきであるからである。すなわち、可能ならば基本ブロック３のスケジューリング結果を用いる。

もし利用可能な情報がない場合、すなわち先に実行されると予測される基本ブロックのスケジューリングが行なわれていない場合には、全実行ユニットが同じタイミングで利用可能になるとしてスケジューリングを行なう。これにより比較的汎用的なスケジューリングが行なえる。

スケジューリングの順番は、多く実行されると予測される基本ブロックを優先する。これにより多く実行される基本ブロックを中心としたスケジューリングが可能となる。この考え方により、図１１のプログラムは、２－３－１－４の順番で基本ブロックのスケジューリングが行なわれる。

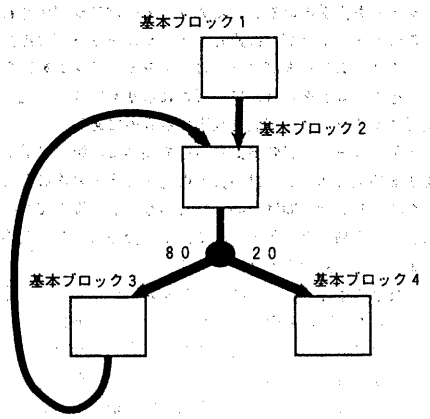


図１１　ループを含んだプログラム構造

５．ダミー方式の性能評価

５．１ 評価方法

ダミー方式の効果をみるために、確率的なモデルでのシミュレーションを行なった。これはある条件のもとに多くのプログラムを確率的に作成し、動的な順序関係をバリア方式とダミー方式を用いて処理し、実行のトレースにより双方の性能を比べるものである。これにより、ダミー方式のバリア方式に対する優位性と、プログラムの特性が変化した場合にどの様にその優位性が変化するかを知ることができる。また、ダミー用に挿入される命令の総数の軽減を考慮することの効果やオーバラップを効率的に行なうためのスケジューリング法の効果もシミュレーションにより評価する。

今回のシミュレーションに用いた条件は以下のようである。

(1) 動的な順序関係の処理方式

動的な順序関係の処理方式として、全実行ユニットがプログラムの流れの制御の時に待ち合わせを行なう基本的なバリア方式、３．２で述べた同期数の削減を行なう拡張を行なったバリア方式、提案手法であるダミー方式の３種類を用いる。そして基本的なバリア方式に対する相対性能として結果の評価を行なう。

(2) プログラム構造

分岐などがどのように行なわれるかを示すものである。無条件分岐や条件分岐を組み合わせて、ｉｆ文のような条件により選択して実行する構造とループの構造を組み合わせたものを用いる。２重ループと選択実行の構造を組み合わせて９個の基本ブロックによって構成されるプログラム構造を用いる。

(3) 実行ユニット数

今回のシミュレーションでは実行ユニット数を５とする。

(4) 命令数

基本ブロック内の命令数は１０～５０で変化させる。これは想定した実行ユニット数を考慮したものであり、実行ユニット数の２～１０倍に設定する。これは現在の１実行ユニットのプログラムにおける１基本ブロック内の命令数が２～１０命令程度であると想定し、その実行ユニット数倍した数である。

(5) 命令の実行時間

プログラム中の命令の実行時間は全て実行前に確定しているものとする。各命令実行時間は１，２，３，４の４種類で、各命令は全て等しい確率で出現するものとしている。

(6) 命令間の順序関係

任意の２命令間の順序関係の存在を確率的に定義する。２つの命令の組み合わせに対して乱数を発生し、ある数を越えた場合にはこの２命令間に順序関係があるものとした。この確率が低い場合には生成されるプログラムの総合的な並列度は高くなり、確率が高い場合にはプログラムの並列度は低くなると考えられる。

この確率と、生成されるプログラムの最大スピードアップとの関係を図１２に示す。最大スピードアップとは、プログラムをシリアルに実行した場合に必要となる実行時間と、並列に実行した場合の最小の実行時間との差を並列で実行した場合の時間で割ったものである。

(7) スケジューリング手法

スケジューリング手法として、ＣＰ法、４．３で述べたダミー命令軽減を考慮したＣＰ法、４．４で述べた拡張を行なったスケジューリング法を用いる。

(8) 計算機の同期機構

計算機の同期機構として一般化静的順序制御機構を用いる。この機構では最も自由な同期による命令間の順序制御が可能であるので、動的な順序関係の各処理方式の最適な場合における性能差を知ることができる。

５．２ シミュレーション結果

結果を図１３，１４，１５に示す。

図１３では横軸が命令間の順序関係の存在確率、縦軸が基本的なバリア方式に対する各方式の平均スピードアップである。

この図ではスケジューリング法にＣＰ法を用いている。図１３の結果から、ほとんどの条件においてダミー方式の性能がバリア方式に対して優れていることが判る。

バリア方式は基本的な方式と、拡張を行なった方式とでは１％未満の性能差しか見られない。これは、バリア方式では、動的な順序関係を保証するために複数の実行ユニットが分岐時に待ち合わせを行なうためである。拡張により同期数を減らした場合でも、相互の待ち合わせによる性能低下の影響により性能はあまり向上されない。それに対して、ダミー方式では最大で３％程度のスピードアップが得られており、ダミー方式の優位性がわかる。順序関係の存在する確率が低い場合に、ダミー方式の性能が拡張を行なったバリア方式の性能とほぼ等しくなっている。これはダミーの順序関係を付加できる命令がプログラム中に存在せず、そのために挿入するダミーの命令の影響が現われていると考えられる。ここではダミーの命令の実行時間は０．５としている。

これに対して、ＣＰ法でダミー命令数の軽減を考慮することによる効果を示すものが図１４である。これよりダミー命令数軽減を考慮しないＣＰ法でスケジューリングする場合に比べて、考慮するＣＰ法でスケジューリングする場合には性能が２％程度改善されることがわかる。

図１５は拡張された提案スケジューリング法の効果を示すものである。ＣＰ法でスケジューリングした場合に対する性能を表している。これより、提案スケジューリング法により４％前後の性能が向上することがわかる。すなわち、提案スケジューリング法により連続して実行される基本ブロック内の命令がより効率良くオーバラップして実行されるようになっていることがわかる。

これらの結果より、ダミー命令の影響を軽減する方法を提案スケジューリング法に応用することにより、効率的な実行が可能になることがわかる。

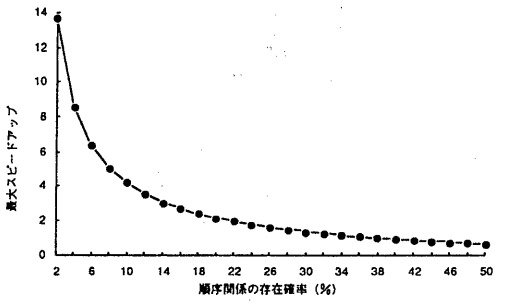


図１２　順序関係の存在確率と生成されるプログラムの特性

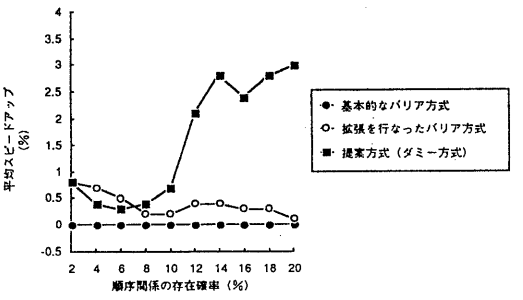


図１３　提案手法の効果

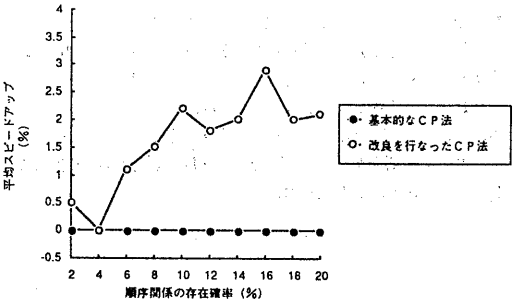


図１４　ＣＰ法の改良による効果

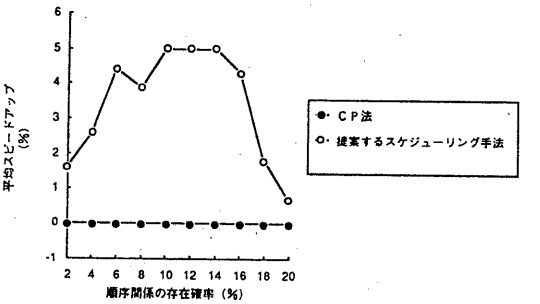


図１５　提案スケジューリング法の効果

６．むすび

静的順序制御機構を用いた並列計算機において、基本ブロック間にまたがる命令間の順序関係を効率的に実現する手法を提案した。本手法を用いることによりプログラム実行に必要となる順序関係を削減することができ、用いない場合に比べて多くの場合で性能を向上することができる。また本手法の特徴を活かすためにはプログラムの流れを考慮したスケジューリングが必要であることを示し、そのようなスケジューリングを行なうことにより、さらに性能を向上させることも可能であることも示した。

本稿では、命令実行時間が変動しないとして評価を行なったが、変動を仮定するとさらに本手法の優位性が高まると考えられる。

参考文献：

1)高木浩光、有田隆也、曽和将容、問題が持つ選考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.1583-1591(1991).

2)R.Gupta,The Fuzzy Barrier:A Mechanism for High Speed Synchronization of Processors, Proc.Third Int.Conf.ASPLOS,pp.54-63(1989).

3)松本尚、Elastic Barrier：一般化されたバリア型同期機構、情報処理学会論文誌、Vol.32,No.7,pp.886-896(1991).
4)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).
5)小島型司、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).