オペレーティング・システム　45 − 7
（１９８9.　１１.　１１）

定並列計算機における
並列ループプログラムの割当

伊藤　勤　　有田　隆也　　曽和　将容
名古屋工業大学　電気情報工学科

従来のマルチプロセッサシステムでは，ループを並列実行させる際に，バリア同期に基づく並列実行方式，あるいはより柔軟なパイプライン実行方式が主に採用されてきた．しかし，パイプライン実行方式においても，ループを構成する各セグメントの実行時間を等しくできない場合，多くの細粒度の待ち時間が発生する．

本研究では，ループ処理の実行過程を解析して，ループ処理の実行中に発生する各プロセッサの細粒度の待ち時間を，その実行の前後に出して他の処理の実行に利用する．これによって実行速度の向上やプロセッサの効率的な利用を図ることができる．本稿では，ループ処理の繰り返し間で限定されたデータ依存関係を持ち，かつ条件分岐を含まないループ本体についての命令割当の一方式を示し簡単に評価する．

Static Scheduling of Loops for Constant-Parallelism Control Flow Computers

Tsutomu Ito　Takaya Arita　Masahiro Sowa

Department of Electrical Engineering and Computer Science
Nagoya Institute of Technology

Gokiso, Nagoya 466, Japan

Barrier synchronization or pipelining is generally used as a synchronization method for parallel execution of loops on the conventional multiprocessor systems. But in pipelining which is more flexible than barrier synchronization, many small busy-waits are liable to be generated if execution time of each segment is not precisely equal. In this research, such small busy-waits can be moved out from the loop execution and utilized for execution of other parts of the program. This paper presents a scheduling algorithm for the deterministic loops which have the restricted data dependency between loop instances. Examples of situations where it can result in improved performance are also presented.

1．はじめに

並列計算機の性能を十分に引き出すためには，素子の高速化はもとより，プログラムの効率的な並列実行が重要である．特にプログラムの実行時間の多くをしめるループ処理の効率的な並列実行は重要である．

一般に，並列処理では，１つのプログラムを複数の処理単位に分割し，それらを複数のプロセッサに割り当てて実行するので，その正しい実行を保証するために同期機構や通信機構が必要になる．特に，ループ処理の並列実行では，繰り返しで副作用が生じないように多くの同期をとるか，繰り返し間でデータを区別する必要がある．

データ駆動計算機におけるループ処理では，副作用が生じないように，動的なデータ管理によって，繰り返し間でデータを区別している[1]．しかし，動的なデータ管理と動的な命令割当によるオーバヘッドが大きい．

一方，マルチプロセッサシステムでは，データは静的に管理され，命令も静的に割り当てられるので，先のようなオーバヘッドはない．このようなシステムにおけるループ処理の並列実行方式として，バリア同期をしながらdoall構文を並列実行する方式がある．これは適用できるループ処理が限定されていて一般的ではない．より柔軟なループ処理の並列実行方式として，パイプライン実行方式がある．

この方式では，ループ本体をセグメントに分割し，ｉ回目の繰り返しにおけるセグメントｓの実行が完全に終了しなければ，ｉ+1回目の繰り返しにおけるセグメントｓの実行が開始できないように同期がとられる．そのようにして，各繰り返しはプロセッサ間でパイプライン的に処理される[2]．この方式ではセグメントの実行時間の差による無駄な待ち時間が発生する．この無駄な待ち時間を小さくするために，セグメントの実行時間を等しくする必要がある．しかし，一般には，多くの無駄な待ち時間が発生する．この待ち時間は細粒度であるため，他の処理の実行にあてようとしても，コンテキストスイッチがオーバヘッドとなり，現実的でない．

以上のようなことを踏まえながら，我々は動的なデータ管理と静的な命令割当による順序制御に基づくシステムを想定し，それにおける並列ループプログラムの命令割当アルゴリズムを提案する．この命令割当アルゴリズムにしたがって命令をプロセッサに割り当てることによって，ループ処理の実行中に発生する各プロセッサの細粒度の待ち時間を，その実行の前後に出す．その結果，その前後に出た時間をひとまとめにして，他の処理の実行に利用することが可能になり，プロセッサの効率的な利用や全実行時間の短縮を図ることができる．本稿では，ループ処理の繰り返し間で限定されたデータ依存関係を持ち，かつ決定的なループ本体についての命令割当アルゴリズムを示し簡単に評価する．

第２章で，提案する命令割当の基本思想について述べ，第３章で，それに基づく並列ループプログラムの命令割当の基本方針について述べる．第４章で，その命令割当アルゴリズムについて述べ，第５章で，本方式について簡単に評価する．

2．命令割当の基本思想

2.1 １つのループ処理に対する命令割当

本命令割当の目的は，ループ処理の実行中に発生する各プロセッサの細粒度の待ち時間を，その実行の前後に出すことである．

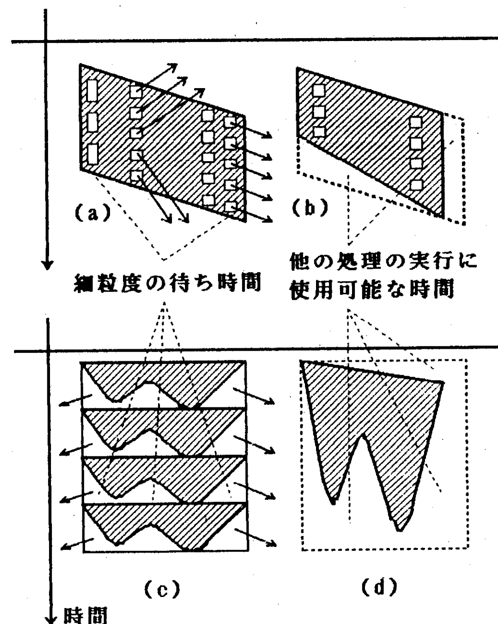


図１．待ち時間の解消

図１は，１つのループ処理における待ち時間の解消を表す概念図である．

(a)はマルチプロセッサシステムにおけるループ処理のパイプライン実行を表している．ここで，四辺形はループ処理の実行を表し，その中の白い四辺形は待ちを表し，斜線の領域は実際の処理の実行を表している．(b)は本システムにおいて，これと同じループ処理を本稿で提案する命令割当アルゴリズムで割当を行った時のその実行を表している．

(a)では，セグメントの実行時間の違いによる細粒度の待ち時間が発生している．理論的には，この細粒度の待ち時間を他の処理の実行にあてることができるが，コンテキストスイッチのオーバヘッドを考えると現実的ではない．

(b)では，動的なデータ管理を行う本システムによって細粒度の待ち時間が自然に解消されている．そして細粒度の待ち時間はループ処理の実行の前後でまとめられ，他の処理の実行にあてることが可能になる．これによって，ループ処理全体の実行時間の短縮やプロセッサ個数の節約が期待できる．

パイプライン実行方式では，各繰り返しを異なるプロセッサに割り当て，各々をオーバーラップさせて実行する．これに対し，ループ本体の並列度が高い場合には，ループ本体を並列実行可能なセグメントに分割し，異なるプロセッサに割り当てて，各繰り返しを複数のプロセッサで処理する実行方式（図１(c)）が考えられる．このような方式においてもセグメントの実行時間の違いによる細粒度の待ちが発生する．この場合も，本方式を適用すると図１(d)のようになり，待ち時間は有効に利用される．

2.2 複数のループ処理の割当

この割当の目的は，2.1 で述べた命令割当によってループ処理の実行の前後に出た時間を，他のループ処理の実行にあてて，全体の実行効率を向上させることである．

図２は，2.1 の割当で使用可能になった時間を他の処理にあてることによる，全体の実行効率の向上を表す概念図である．ここでは，プロセッサの個数が限られているシステム（たとえば[3]など）で，３つのループ処理（処理ＡとＢは並列実行可能で，処理Ｃは処理Ａ,Ｂの後にしか実行できないとする）を実行することを考える．同図(a)はマルチプロセッサシステムに対する従来の一割当方法による，複数のループ処理の実行を表している．同図(b)は本システムにおける複数のループ処理の割当方法による実行を表している．

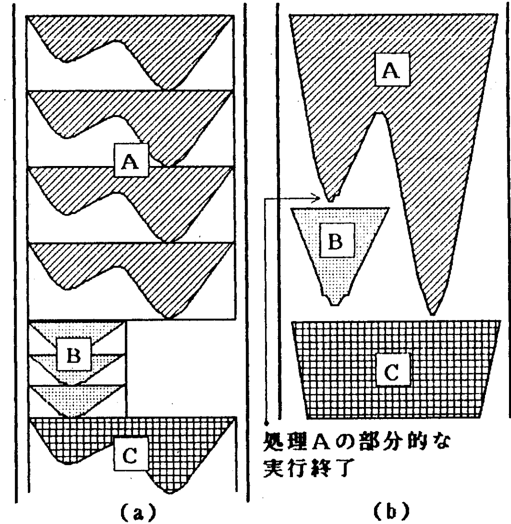


図２．使用可能になった時間の積極的な使用による実行効率の向上

プロセッサの数が限られているため，(a)では処理ＡとＢを並列に実行することができず，結局３つの処理を逐次的に実行しなければならない．一方(b)では処理Ａの部分的な実行終了後，処理Ｂの実行を開始する．その結果，(a)に比べて早い時刻に処理Ｃの実行を開始できる．

このようにして，１つ１つのループ処理の実行中に発生する待ち時間をその実行の前後に出して，それを積極的に使うことによって，全体の実行効率を向上させ，プログラム全体の実行時間を短縮することができる．

３．命令割当アルゴリズムの基本方針

2.1 で述べた１つのループ処理に対する命令割当に基づくアルゴリズムの基本方針について述べる．2.2 の複数のループ処理の割当アルゴリズムについては本稿では詳しく触れない．

3.1 前提

本アルゴリズムが対象とするループ処理と，想定するシステムに関する前提は次の通りである．

(1) 対象とするループ処理に関する前提

① 繰り返し間のデータ依存関係

ループ処理の繰り返し間では，i回目の繰り返しで定義されるデータは，i+1回目の繰り返しで使用される．

② ループ本体

条件分岐命令などの非決定的な命令や，ループ処理を含まない．

(2) 想定するシステムに関する前提

① 動的なデータ管理

命令の実行によって生成されるデータは，それを必要とする命令に直接"値"として渡される．また，ループ処理における変数は，実行時に，繰り返しごとに区別される．

② 静的な命令割当

③ プロセッサ間のデータ送受信の時間は無視

3.2 繰り返しのオーバラップ

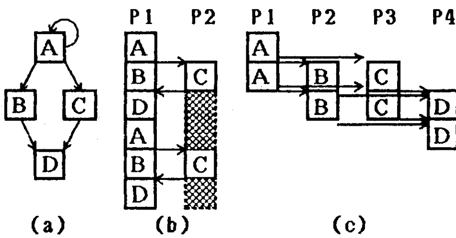


図3．繰り返しのオーバラップによる実行時間の短縮

第1の方針は，各繰り返しをオーバラップさせて実行することにより，1つのループ処理全体の実行時間をより小さくすることである．

(a)は簡単なデータフロープログラムである．四角形は命令を表し，矢印はデータの依存関係を表し，網目の領域は待ち時間を表している．また命令の実行時間は一定とする．命令Aは自ら生成したデータを消費して次のデータを生成する．また命令Aは自ら生成したデータを命令B,Cに次々に送る．命令B,Cは命令Aから送られてくるデータを順番に処理しながら，その結果を次々に命令Dに送る．命令Dは命令B,Cから送られてくるデータを順番に処理する．

(b)は繰り返しのオーバラップについて考慮せず，命令A,B,CをプロセッサP1に，命令DをプロセッサP2に割り当てて実行させたものである．繰り返しのオーバラップはできていない．

命令Aはそれ自身とのデータ依存関係しかないので，命令Aは連続的に実行可能である．したがって，命令Aは1回目の実行後，すぐに2回目の実行を開始することができる．一方，命令Aの1回目の実行後，命令B,Cは1回目の実行を開始することができる．すなわち，2回目の命令Aの実行と1回目の命令B,Cの実行をオーバラップさせて並列に行うことができる．したがって，このような繰り返しのオーバラップを実現するには，命令A,B,Cは異なるプロセッサに割り当てればよい．

(c)はこのような繰り返しのオーバラップのために命令A,B,C,DをプロセッサP1,P2,P3,P4に割り当てて実行させたものである．繰り返しをオーバラップさせることによって，実行時間が短縮された．

3.3 データ送受信の排除

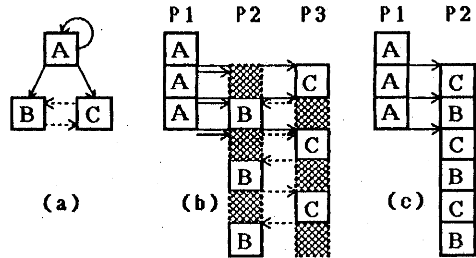


図4．相互データ通信の排除

第2の方針は，データ送受信を排除することにより，同期のオーバヘッドを解消したり，使用するプロセッサの数を節約することである．

図4のプログラム(a)は，命令B,Cは互いにデータを送受信しながら実行される．いま命令Aは連続して実行され，その結果が命令B,Cに次々に送られたとする．命令Bは命令Cからのデータを待ち，命令Cは命令Bからのデータを待つのでデッドロックとなる．ここで命令Cの実行に必要な初期データが与えられ，1回目の命令Cが実行されたとすると，1回目の命令Bが実行され，次に2回目の命令Cが実行される．このようにして命令B,Cの実行が進む．

(b)は，第1の方針にしたがい，繰り返しの最適なオーバラップを期待して，命令A,B,CをプロセッサP1,P2,P3に割り当てて実行させたものである．これはデ

ータ送受信を考慮しなかったので，命令Ｂ，Ｃは同時に実行できないにもかかわらず異なるプロセッサに割り当てられている．したがって，データ送受信の同期による待ち時間の発生を招いている．このように，この例ではデータ送受信は本質的に逐次的なので，それらは１個のプロセッサに割り当てる方がよい．データ送受信を持つ命令Ｂ，ＣをプロセッサP2に割り当てて実行させると(c)のようになり，同期による待ち時間が解消され，プロセッサの使用個数が１減る．

3.4 命令の埋め込み

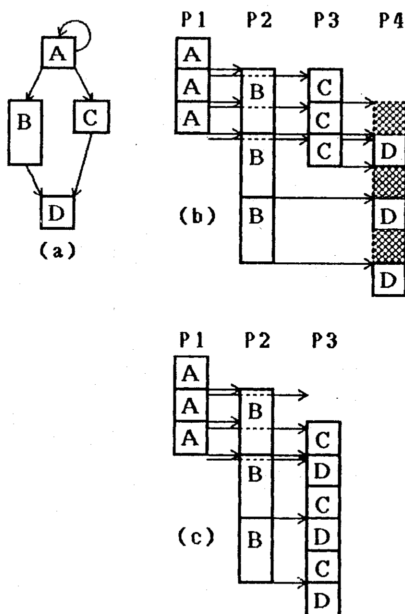


図５．適切な命令合成

第３の方針は，命令を埋め込むことにより，待ち時間を解消することである．

図５のプログラム(a)で，命令Ｂの実行時間は，他の命令の２倍とする．グラフの形は，図３(a)と同じである．

(b)は，第１の方針にしたがい，繰り返しの最適なオーバラップを期待して，命令Ａ，Ｂ，Ｃ，ＤをプロセッサP1,P2,P3,P4に割り当てて実行させたものである．命令Ｂの実行時間が変わったことにより，プロセッサP4で待ち時間が生じている．この待ち時間を解消する手段として，これを他の命令で埋めることが考えられる．

命令Ａ，Ｂ，Ｄを通るパスは，繰り返しの実行の中で最も時間のかかるパスであるから，これらの命令実行の遅延はループ処理全体の実行時間を長くする．したがって，ループ処理全体の実行時間を長くしたくない場合は，これらの命令でその待ち時間を埋めることはできない．一方，命令Ｃは命令Ａの実行終了後に実行可能になり，命令Ｄの実行開始までにその実行を終了すればよい．すなわち命令Ｃを命令Ｄの間に埋め込み，プロセッサP3に割り当てる．その結果(c)のようになり，待ち時間が解消され，プロセッサの使用個数が１減る．

4．命令割当アルゴリズム

第３章で述べた基本方針に基づいた命令割当アルゴリズムについて述べる．

4.1 命令割当アルゴリズム

1） 問題をデータフローグラフで表現する．ループ本体を表すグラフにおけるパスのうち，最も実行時間の長いパス（クリティカルパス）を見つける．

2） クリティカルパス上の各命令に対して各々１個のプロセッサを割り当て，それらをそれらの先行順序にしたがって並べる．これをｋ回目の繰り返しの実行とみなす．

3） 2)で得られたｋ回目の繰り返しを表す並びの後に，k+1回目の繰り返しを表す並びを書く．次に，それによって生じる待ち時間をクリティカルパス上の命令で埋めあう．

4） 割り当てられずに残っている各命令の実行に対して，E-level（Earliest Execution Level：その命令が実行できる最も早い時刻）とL-level（Latest Execution Level：クリティカルパス全体の実行時間を延ばさずにその命令を実行できる最も遅い時刻）を求める[4]．

5） L-levelが最も遅いものを選び，それをE-levelからL-levelまで動かした時に，その命令の実行時間よりも長い待ち時間があればそこにその命令を挿入する，なければその命令に１個のプロセッサを割り当てる．

6) 割り当てられずに残っている命令の実行に対して，4),5)を繰り返す．

4.2 命令割当

命令割当アルゴリズムにしたがって，図6に示すサンプルプログラムの命令割当を行う．ただし，図6において命令の右の数字は，その命令の実行時間を表している．

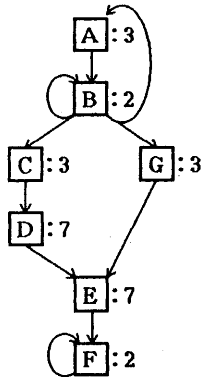


図6．サンプルプログラム

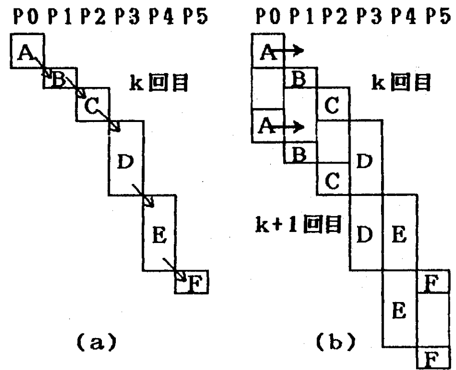
1) ループ本体を表すグラフにおけるクリティカルパスは命令A，B，C，D，E，Fである．

2) 命令A，B，C，D，E，Fを，プロセッサP0,P1,P2,P3,P4,P5に割り当てる．これらの命令を，先行関係A→B→C→D→E→Fにしたがって並べると(a)のようになる．これをk回目の繰り返しの実行とみなす．

3) 2)で得られたk回目の繰り返しを表す並びの後に，k+1回目の繰り返しを表す並びを書くと(b)のようになる．次に，命令Bのk-1回目の実行とk回目の実行の間の待ち時間を命令Aのk回目の実行で埋め，命令Bのk回目の実行とk+1回目の実行の間の待ち時間を命令Aのk+1回目の実行で埋める．

4) 割り当てられずに残っている命令Gのk回目の実行は，命令Bのk回目の実行後のデータを受け取って実行可能になる．また，それは命令Eのk回目の実行が開始されるまでにその実行を終了していればよい．したがって，命令Gのk回目の実行に対するE-level，L-levelは(c)のようになる．破線の長方形は，命令Gが全体の実行時間を延ばさずに移動できる範囲を示している．命令Gをこの範囲で移動してみると，それは命令Cのk回目の実行とk+1回目の実行の間に挿入することができるので，命令GはP2に割り当てる．

以上の過程を経て，P1には命令AとB，P2には命令CとG，P3には命令D，P4には命令E，P5には命令Fが割り当てて実行すると(d)のようになる．



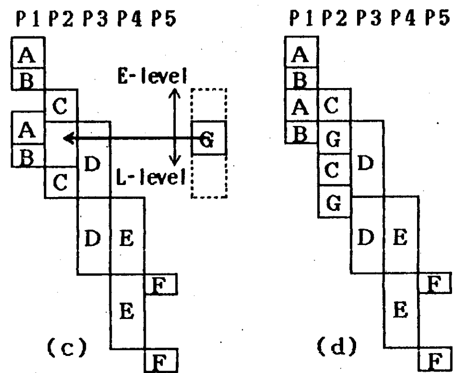


図7．命令割当の過程

5．簡単な評価

本実行方式を，ループ処理の各繰り返しをオーバラップさせて実行するという点が本実行方式に似ているパイプライン実行方式と簡単に比較し検討する．

比較する項目は，ループ処理の実行時間とループ処理実行中のプロセッサ稼働率とする．ここで実行時間とは，そのループ処理の実行が開始されてから，全処理が完全に終了するまでの時間である．稼働率は次の式で定義する．

$$\text{稼働率} = \frac{E}{M} = \frac{\Sigma\ Ei}{\Sigma\ Mi}$$

ただし，Eiはプロセッサｉが自分に割り当てられたループ処理を開始してから終了するまでに実行した命令

の実行時間の合計を表し，Miはプロセッサｉが自分に割り当てられたループ処理を開始してから終了するまでの時間を表す．稼働時間ＥはＥ＝ΣＥｉ，専有時間ＭはＭ＝ΣＭｉである．

比較のために用いたサンプルプログラムは，内積計算(scalar)，台形公式を用いた積分計算(trape)，シンプソン公式を用いた積分計算(Simp)である．

各命令のの実行時間は，繰り返し判定命令３，加減算２，乗算７とする．またプロセッサ間のデータ送受信のために必要な時間は無視できるものとする．

パイプライン実行方式では，各ステージで処理されるセグメントの実行時間をできるだけ等しくなるように最適化して実行させた．本実行方式では，第４章の命令割当アルゴリズムにしたがって，想定するシステムに命令を割り当てて実行させた．その結果を表１,２に示す．ただし，ｎは繰り返しの回数を表している．

表１．パイプライン実行方式による実行結果

	実行時間	専有時間	稼働時間	稼働率
scalar	7n +28	35n	27n	77.1 %
trape	7n +42	49n	39n	79.6 %
Simp	7n +126	133n	96n	72.2 %

表２．本実行方式による実行結果

	実行時間	専有時間	稼働時間	稼働率
scalar	7n +17	32n -5	27n	84.4 %
trape	7n +25	47n -8	39n	83.0 %
Simp	7n +41	110n -14	96n	83.3 %

この例では，両方式とも実行時間の差はほとんどないが，稼働率は，本方式の方がパイプライン実行方式より５から10%高いことが示された．この稼働率の差は，ループ処理の内部からその前後に出すことができた（使用可能となった）待ち時間の量を表している．

稼働率は100%には至らないが，これはプログラムに含まれる本質的な待ち時間によるものであり，それ以外の待ち時間はほとんど処理の前後に出すことができたと思われる．

６．おわりに

ループの並列実行の際に生じる細粒度の待ち時間を，その実行の前後に出して他の処理の実行に利用することにより，実行速度の向上やプロセッサの効率的な利用を図る命令割当アルゴリズムを示した．そしてパイプライン方式との比較を行い，その有効性を示した．

本稿ではプロセッサ間のデータ送受信時間は無視したが，プロセッサが多い場合などには，プロセッサ間のデータ送受信を考慮したアルゴリズムについても検討する必要がある．また本アルゴリズムが適用可能なループ処理は限られているので，より一般的なループ処理に適用可能なアルゴリズムについての検討も課題である．

参考文献

[1] Arvind, and Gostelow,K.P.:The U-interpreter, IEEE Computer, Vol.15,No.2, pp.42-48(1982).

[2] Michael Wolfe : Multiprocessor Synchronization for Concurrent Loops, IEEE Software, Jan, pp.34-42(1988).

[3] 曽和将容，有田隆也：コンスタントパラレリズム・コントロールフロー計算機の提案，信学技報，Vol.89, No.55, pp.1-7(1989).

[4] M.Tokoro,J.R.Jagannathan,and H.Sunahara.: On the Working Set Concept for Data-flow Machines, Proc.of the 10th Annual Symposium on Computer Architecture, pp.90-97(1983).