並列処理システムにおける
実行制御方式に関する研究

1990年12月

小　林　真　也

並列処理システムにおける
実行制御方式に関する研究

1990年12月

小　林　真　也

内容梗概

本論文は，著者が大阪大学大学院工学研究科（通信工学専攻）在学中に行った研究の成果をまとめたものである．全文は7章から構成されている．

第1章は緒論であり，本研究の目的と技術的背景について述べる．

第2章は，並列処理システムと並列処理する際に問題となるスケジュール不可能なジョブについて述べる．並列処理システムを，ジョブの投入から処理結果の出力までを考慮すると，逐次処理にはなかった処理として，ジョブのタスクへの分割，プロセッサへのタスクの割当，プロセッサ間の同期の3つの処理が必要となる．本章では，これらの処理に要求される機能について述べる．また，処理過程が処理の開始以前に決定されていないスケジュール不可能なジョブについて述べ，スケジュール不可能なジョブを並列処理システムで処理する際の問題点を明らかにする．

第3章では，従来の逐次プログラミング言語で記述された，ジョブを並列処理システムで実行するために，タスク間のメッセージの送受信をプログラム中から検出しそれとともにジョブをタスクに分割するメッセージ依存分割法と，得られたタスクの依存関係検出法の提案を行う．メッセージ依存分割法はプログラム中からメッセージの送受信点を検出し，その送受信点で分割を行うことにより，タスクを生成する分割法である．このようにして得られたタスクは内部手続きと内部状態を持ち，オブジェクト指向のオブジェクトのように，互いにメッセージの送受信によって関係しあい，メッセージの受信により起動し，終了時に他のタスクに対してメッセージを送信する．さらに本章ではメッセージ依存分割法と従来の分割法との比較検討を行い，従来のジョブ分割法に比べ，より高い並列性を抽出でき，また同程度の並列性を抽出する際には分割回数が少ないことを示している．また，従来の分割法では分割できなかったスケジュール不可能なジョブに対しても分割を行うことができることを示している．

第4章では，タスクをプロセッサに割当てるタスク割当法として，タスク多重割当法の提案を行う．本方式は，発生条件が異なるために，実

際の処理系列において共に発生することのない互いに排反なタスクを同一のプロセッサに割当てる（多重割当て）ことを可能とする方式である．多重割当てされたタスクは発生条件が異なるため，実際の処理系列においてプロセッサの競合を起こすことはない．さらに本章では，本方式とタスクの排反性を考慮しない割当法との比較検討を行い，タスク多重割当法の方が，同程度の処理終了時間を得るために必要となるプロセッサ台数が少ないことを，また，プロセッサ台数に制限がある場合にはより短い処理終了時間となることを示している．

第5章では，スケジュール不可能なジョブを効率良く処理する目的で提案されたプロセッサ間同期方式である先行制御方式を対象に理論解析を行う．プロセッサ間同期方式の解析を行うために，プロセッサ間の通信をモデル化する方法として，拡張メッセージ・パッシングの提案を行う．拡張メッセージ・パッシングは，確率的なメッセージの送信確率と送受信時の時刻の分布を取り扱えるように，従来のメッセージ・パッシングを拡張したものである．この拡張メッセージ・パッシングを用いて，先行制御方式を用いた場合のプロセッサ間関係をモデル化し，先行制御方式おいてオーバヘッドの主な原因となる矛盾状態の発生率を理論解析により求めている．また，この解析結果をもとに，先行制御方式を用いた並列処理システムでは実処理速度改善比を，送信側に小規模なジョブを多く，受信側に大規模なジョブを少なく割当て負荷を均一化することで，改善できることを示している．

第6章は，並列処理システムの応用例として分散データベースを取り上げ，分散データベースの同時実行制御方式として，不変時刻印方式を提案し，その正当性を証明する．分散データベースの首尾一貫性を保証する同時実行制御方式として時刻印方式が従来から提案されてきた．しかしながら，従来の時刻印方式では要求の受付順にサービスを行うことは保証されず，公平性が損なわれる．本論文で提案する不変時刻印方式は，データ書き込み操作ごとに作成される版に対してすべての読み出し操作の記録を取ることにより，読み出し操作，書き込み操作ともにアボートされることなく受付順にサービスを行うことができ，優先順位のあるサービスも実現できる同時実行制御方式である．

第7章は結論であり，本研究で得られた諸結果をまとめ，若干の検討を加える．

関係発表論文

I. 学会論文

[1] 小林，渡辺，中西，手塚：“拡張メッセージ・パッシングを用いた先行制御方式のモデル化と解析”電子情報通信学会D-I(採録決定)

[2] 小林，古川，中西，手塚：“要求順サービス可能な時刻印方式について”，電子情報通信学会D-I(採録決定)

II. 学会講演発表

[1] 小林，渡辺，真田，手塚：“メッセージパッシングによる並行処理系のプロセス間関係の取り扱いについて”，昭和63年電子情報通信学会春季全大D-380(1988-3)

[2] 小林，渡辺，中西，手塚：“先行制御方式におけるオーバヘッドの解析”，情報処理学会第37回(昭和63年後期)全大5E-10(1988-9)

[3] 古川，小林，中西，手塚：“シミュレーションによる先行制御方式の処理効率解析”，昭和63年電気関係学会関西支部連合大会，S7-8(1988-11)

[4] 古川，小林，中西，手塚：“要求順サービスを可能とする不変時刻印方式の提案”，1990年電子情報通信学会春季全大D-73(1990-3)

[5] 小林，中西，手塚：“タスク多重割当法の提案”1990年電子情報通信学会秋季全大D-56(1990-10)

[6] 小林，村田，中西，手塚：“オブジェクト指向並列処理システムの提案”，1990年電子情報通信学会秋季全大D-98(1990-10)

III. 研究会発表

[1] 小林，渡辺，真田，手塚：“確率的要素を取り扱うためのメッセージパッシングの拡張”，電子情報通信学会コンピュテーション研究会，COMP87-59(1987-12)

[2] 古川，小林，中西，手塚：“先行制御方式を用いた並行処理システムの性能評価”，電子情報通信学会システムのモデリングと性能評価時限研究専門委員会資料，(1988-6)

[3] 小林，渡辺，中西，手塚：“並行処理システムにおけるプロセッサ間同期方式について”，電子情報通信学会交換システム研究会，SSE88-82((1988-7)

[4] 村田，小林，中西，手塚：“プログラム構成単位間通信に注目した並列性抽出に関する研究”，電子情報通信学会コンピュータシステム研究会，CPSY89-58(1989-8)

[5] 古川，小林，中西，手塚：“要求順サービスを可能とする制御方式について”，電子情報通信学会データベースワークショップ資料，p121-125(1990-1)

[6] 村田，小林，中西，手塚：“並列実行性に着目したプログラム分割と構造解析”情報処理学会計算機アーキテクチャ研究会，ARC83-9(1990-7)

[7] 小林，小巻，中西，手塚：“タスク間の排反性に注目したタスク割当法”，電子情報通信学会コンピュータシステム研究会，CPSY90-34(1990-7)

目 次

第1章

緒論

高度情報化社会を迎えた今日において，ジョブを高速に処理する計算機システム構築の要求が高まっている．高速な計算機システムを実現する手法の一つに，並列処理 [KOCK 81][AISO 82][HWAN 84][COMP 85][SAIT 89][TAKA 89][TOMI 89]がある．並列処理は処理装置を複数台接続し処理を行うもので，処理の高速化のみならず，処理装置が複数台接続されているため，システムの一部に故障が発生しても処理を継続することができ，高い信頼性を得ることができる．また，処理装置の追加・削除が容易に行え，高い拡張性をも得ることができる．

並列処理システムの実現性が高まってきた背景として，まず第一に半導体技術の発展による，マイクロプロセッサなどの安価なプロセッサの出現により複数のプロセッサを用いたマルチプロセッサシステムの実用化の条件が整ってきたことがあげられる．またコンピュータ・ネットワークの出現により，従来のコンピュータ相互の1対1の通信形態から，複数のコンピュータを有機的に接続利用するマルチコンピュータ環境の出現があげられる．

現在実用化されている並列処理システムは画像処理やシミュレーションなどの特定のジョブに対する専用システムや，ベクトルプロセッサのようにＤＯループなどのプログラムの一部から並列性を抽出し，並列処理を行うものであり，従来からの逐次プログラミング言語で記述されたジョブ全体から並列性を抽出し，並列処理する汎用並列処理システム [KOBA1 90c]は実用化されていない．汎用並列処理システム（以下特に区

別の必要のない限り，並列処理システムと呼ぶ）においては逐次処理システムでは必要のなかった実行制御が必要となる．この実行制御方式には，ジョブ分割法，タスク割当法，プロセッサ間同期法の３つがある．ジョブ分割法は，システムで処理されるジョブを並列処理可能な複数のタスクに分割するものであり，ジョブ分割によって得られたタスクをプロセッサに割当てるのがタスク割当法である．また，プロセッサ間同期法は，処理結果に矛盾が生じないように，各プロセッサの処理の進行・停止を決定し同期をとるものである．

本研究は，逐次プログラミング言語で記述されたジョブから，最大限の並列性を抽出し並列処理する汎用並列処理システムを対象に，その実現に不可欠な実行制御方式であるジョブ分割法，タスク割当法，およびプロセッサ間同期法について方式の提案ならびに種々の考察を行ったものである．

本論文では，まずプログラム構成単位間の通信に注目しジョブの分割を行うメッセージ依存分割法の提案を行う．また，実際の処理系列においてともに発生することのない互いに排反なタスクを同一のプロセッサに割当てるタスク多重割当法の提案を行う．次に，プロセッサ間同期法である先行制御方式においてオーバヘッドの主な原因となる矛盾状態の発生率を理論解析によって求め，高速処理を行うためのタスク割当ての条件を示す．さらに，並列処理システムの応用として，分散データベースを取り上げ，データベースの首尾一貫性を保証する同時実行制御方式として，受付順サービス可能な不変時刻印方式の提案を行う．

まず，逐次プログラミング言語で記述されたジョブに対して，プログラムを構成するプログラム構成単位間の依存関係に注目し分割を行うメッセージ依存ジョブ分割法[MURA1 89][MURA1 90]の提案を行う．メッセージ依存分割法では，プログラム構成単位間の通信が行われるところをプログラムから検出し分割を行う．この結果，処理過程が処理の開始以前に決定していないため，従来の分割法では取り扱うことができなかったスケジュール不可能なジョブの分割も行うことができる．また，メッセージ依存ジョブ分割法によって得られるタスクはオブジェクト指向[SHIB 88]におけるオブジェクトとして振る舞うため，処理の実行時に従来のデータ依存関係のうち逆依存と出力依存とは考慮することなく，フロー依存関係にのみ矛盾しないように同期をとれば良い．

また，並列処理による処理過程に注目すると，タスクの中にはともに発生する処理系列が存在しないタスクの組がある．これをタスクの排反関係と呼び，これらのタスクを互いに排反なタスクと呼ぶ．互いに排反なタスクは，ともに発生する処理系列が存在しないため，これらのタスクを同一のプロセッサに割当てても，処理実行時にプロセッサの利用を巡り競合することはない．また，一方のタスクが発生しないときに，他方のタスクが発生すればプロセッサが何も処理をしないアイドル状態とはならない．そこで，本研究では，互いに排反なタスクを同一のプロセッサに割当てる多重割当てを行うタスク多重割当法の提案を行う．さらに，このタスク多重割当法が従来の割当法に比べ，処理終了時間と必要プロセッサ台数の点で優れていることを示す．

次に，本研究ではスケジュール不可能なジョブを並列処理する際のプロセッサ間同期法として提案されてきた先行制御方式を対象に理論解析を行う．プロセッサ間同期法を理論解析するためには．プロセッサ間の関係をモデル化する必要がある．本研究では，このモデル化手法として，従来オブジェクト指向におけるオブジェクト間の通信を取り扱う概念として用いられてきたメッセージ・パッシングに対して，確率的な要素を取り扱えるように拡張した拡張メッセージ・パッシングの提案を行う．さらに，先行制御方式における処理速度低下の主な原因となる矛盾状態の発生に注目し，矛盾状態発生率を理論解析により求め，これをもとに，同期によるオーバヘッドを少なくできる割当てを行うための条件を示す．

さらに，本研究では並列処理システムの応用として，分散データベース[KAMB 86][NISH 90]を取り上げる．分散データベースは複数のデータベースをネットワークを介して接続し，利用者からはあたかも単一のデータベースの様に利用できるシステムである[MASU 87]．このような分散データベースでは，データの首尾一貫性を守るために同時実行制御方式が必要となる．従来から提案されていた同時実行制御方式には2相ロック方式[ESWA 76][KOHL 81][KAMB 86][KOBA2 86][KAMB 87a][KAMB 87c][KAMB 88]や時刻印方式[BERN 81][ZHON 86][KAMB 87b][ZHON 89]がある．しかしながら，これらの同時実行制御方式では，利用者からの処理要求であるトランザクションを受付順に処理することができなかった．そこで，本研究では，データ項目に対する書き込み・読み出しの履歴を保存することで，

トランザクションの受付順サービスを可能とする不変時刻印方式の提案を行う．さらに，不変時刻印方式が，同時実行制御方式が満たすべき条件である直列可能性を満たすと同時に，デッドロックやロックアウトが生じない方式であることを証明する．

第2章

並列処理システム

2.1　緒言

並列処理技術は，大規模なジョブを高速に処理する方法として注目され，画像処理[MITSU 85][KAWA 88][TADA 88]や行列計算等の分野において広く用いられている．

本章では，まずジョブを処理の実行過程が処理の開始以前に決定されているスケジュール可能なジョブと，処理の実行にともなって処理過程が決定されるスケジュール不可能なジョブの2つに分類する．

つぎに，並列処理システムで処理を行う際に必要となる実行制御方式について述べる．並列処理システムの実行制御方式はジョブ分割法・タスク割当法・プロセッサ間同期法の3つがある．

ジョブ分割法は，ジョブを並列処理可能なタスクに分割する役割を担っている．また，タスク割当法は得られたタスク集合の持つ並列性を損なうことなくプロセッサに割当てる役割を果たす．一方，プロセッサ間同期法は，処理結果に矛盾がないようにプロセッサの処理の実行・停止を制御するものである．

本章では，これら3つの実行制御方式について要求される機能，特徴について詳説する．

2.2　ジョブの分類

ソフトウェアの面から捉えると，処理システムで実行されるジョブはいくつかのタスクに分割され，これらはいくつかのプロセスに割当てられる．プロセスとは抽象的な機能単位であり，プロセスの数や種類は受け持つタスクの内容によって決定される．たとえば，ジョブの一例として待ち行列網シミュレーションを考えると，各ノードをプロセスに対応させたり，マンマシンインターフェイス部や待ち行列の管理部などをプロセスに対応させることができる．各プロセスは他のプロセスとメッセージの送受信によって互いに関係を持ちながらタスクを実行する．このタスクには，

① 他のプロセスへのメッセージの送信の処理
② 他のプロセスから受信したメッセージによって要求される処理
③ 他のプロセスとは無関係な処理

がある．

これらのタスクの実行をイベントと呼ぶ．イベントの実行順序には半順序関係があり，タスクはこの順序関係を乱すことなく実行されなければならない．

また，処理システムにおいてプロセスに割当てられたタスクを実際に処理するのはプロセッサである．プロセッサにはいくつかのプロセスが割当てられ，各プロセッサは割当てられたプロセスが行うべきタスクの処理を実際に行う．

本論文では，処理システムにジョブが与えられたときに生成されるプロセス数nはプロセッサ数mに等しい（m＝n），つまり1台のプロセッサが1つのプロセスを受け持つと仮定する．n＞mの場合には1台のプロセッサに2つ以上のプロセスが割当てられるが，これら同一のプロセッサに割当てられたプロセスを同じ働きをする1つのプロセスとみなすことによりn＝mの場合に近似することができる．以下，特に区別の必要がない限り本論文ではプロセスの意味も含めてプロセッサと呼ぶことにする．

並列処理システムでの種々のジョブの実行を考えると，画像処理のよ

うなジョブでは，各プロセッサにおけるイベントの順序がジョブの実行以前に決定されている．すなわち，各プロセッサにおけるイベントの発生順序は処理の開始以前に（これを“予め”と呼ぶ）決定できる．これに対し，待ち行列網シミュレーションや分散データベースのように各プロセッサにおけるイベントの発生が処理にともなって決定されていくようなジョブでは，プロセッサ間の同期をいつ行うかは処理の開始以前には決定できない．このような観点から，待ち行列網シミュレーション等をスケジュール不可能なジョブ，前述の画像処理等をスケジュール可能なジョブと呼ぶことができる．以下にスケジュール可能なジョブとスケジュール不可能なジョブにおけるイベントの発生の順序関係の例を示す．なお，各イベントを［＊］で表現する．

(1)　スケジュール可能なジョブ

イベント間の半順序関係が確定しているジョブの例としては，図2-1に示すようなイベントの発生順序関係を持つジョブが考えられる．

(2)　スケジュール不可能なジョブ

スケジュール不可能なジョブの具体的な例を図2-2に示す．この例では，イベント［1］が発生した後イベント［11］が発生するか［12］が発生するかが確率的に決定され，［11］が発生した場合は［22］が発生し，［12］が発生した場合は［21］が発生する．この場

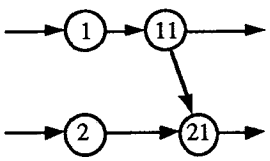


図2-1　スケジュール可能なジョブ

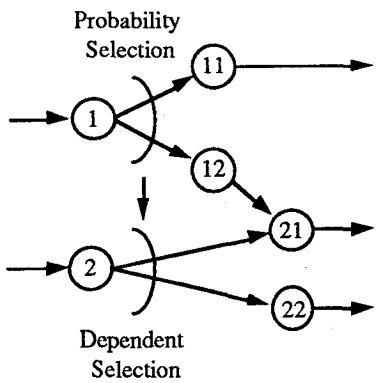


図2-2　スケジュール不可能なジョブ

合，イベント［1］が終了するまで［11］と［12］の何れが発生するか確定していない．このようなジョブの実行に当っては，実行開始以前にイベントの種類とその発生順序は不明である．

2.3　並列処理システムの実行制御方式

並列処理システムの処理過程においては，ジョブ分割法・タスク割当法・プロセッサ間同期法の3つの実行制御方式が必要となる．これら3つの実行制御方式は逐次処理システムで処理をする際には，必要とならないが，並列処理システムの性能は，これら3つの実行制御方式に大きく依存する．以下にこれら3つの実行制御方式の役割と要求される機能について述べる．

2.3.1　ジョブ分割法

ジョブの持つ並列性を利用し，処理を高速に行う並列処理システムでは，ジョブの持つ並列性の抽出が重要な課題となる．ジョブの持つ並列性を抽出する方法としては，利用者が並行プログラミング言語を用いてジョブの持つ並列性を明示的に記述する方法[OOMO 90][BENA 82]と従来の逐次プログラミング言語でジョブが記述されたプログラムからジョブ分割法により並列性を抽出する方法[MURA2 90]の2通りがある．前者の並行プログラミング言語による記述では，利用者がジョブの並列性の抽出・記述を行うので，処理を行う際に並列性抽出のためのオーバヘッドを発生しないという利点がある．しかしながら，この方法による並列性抽出は，ジョブを記述する利用者に委ねられているため，利用者の並列性抽出・記述能力に大きく依存し，ジョブの記述には熟練を要し，初心者が容易に並列性を記述することができないという欠点を有する．また，ジョブの並列性抽出は，利用者が記述したレベルに限られ，ジョブの持つ潜在的な並列性を全て利用することができない．これに対し，後者のジョブ分割法による並列性抽出は，並列性抽出のためのオーバヘッドが必要となるものの，利用者に対して並列性抽出に関し何等負担を負わせることがなく，利用者のプログラミング能力に関係なく並列性を抽出できる．また，従来の逐次プログラミング言語によって記述されたジョブ

を対象とするため，ジョブを新たに並行プログラミング言語で記述しなおすことなく，逐次プログラミング言語で記述されたプログラムの蓄積を利用することができる．そこで，本論文では，このジョブ分割法による並列性抽出について検討を行う．

並列処理システムにおけるジョブ分割は，プログラミング言語で記述されたジョブを並列処理可能な複数個のタスクに分割することを意味する．このジョブ分割を行う際に要求される特徴としては以下に示す様なものがある．

・効率の良い並列性の抽出

分割を行うことにより，ジョブの並列実行の可能性を最大限に引き出す．その際，並列処理システムでの高速な処理を妨げる原因となる分割時間，分割回数，並列実行のために各タスクに付加されるオーバヘッドなどを最小限に押える必要がある．

・タスク特性の抽出

ジョブ分割することによって得られたタスクをプロセッサに割当てる際や，タスクの実行を行う際に必要となる情報を得るために，プログラムの構造を解析し，各タスクの特性を検出する必要がある．

2.3.2　タスク割当法

並列処理システムでジョブの実行を行うためには，ジョブ分割法によって得られたタスクを，各プロセッサに割当てる必要がある．高速な処理を実現するためには，タスク割当法に要求される機能として以下に示すものがある．

・プロセッサの負荷の均一化

並列処理システムでジョブを実行する際には，システム全体としての処理終了時間は，最も処理終了時間の遅いプロセスの終了時間となる．したがって，負荷が不均一に割当てられていれば，高負荷のプロセッサの処理終了時間が遅くなり，ジョブ終了時間が遅くなる．そこで，並列処理システムでは，各プロセッサの負荷が均一となるように割当てを行う必要がある．

・ジョブ分割法によって得られた並列性を損なわない．

並列実行可能なタスクを同一のプロセッサに割当てると，逐次処理されるため並列性が損なわれる．このように並列性を損なわないように，並列実行可能なタスクを異なるプロセッサに割当てる必要がある．

・通信によるオーバヘッドの低減

並列処理システムで実行されるタスク間には依存関係があり，このタスク間の依存関係は，ジョブの実行時にはプロセッサ間の通信を引き起こす．このプロセッサ間の通信は本来ジョブにはなかった処理であり，オーバヘッドとなる．したがって，タスクをプロセッサに割当てる際には．この通信によるオーバヘッドが少なくなるように割当てを行わなければならない．

次に，タスク割当法を割当て処理が行われる時点によって，以下の3種類に分類する．

（1）初期割当て

初期割当ては，並列処理システムがジョブの処理を開始する以前に行われる割当てであり，並列処理を行う際の初期効率は初期割当ての性能に依存する．初期割当てにおいては，ジョブ分割法から与えられるタスクの個々の処理や依存関係などの情報や，プロセッサ間の通信に要するオーバヘッドなどの情報を用いて割当てを行う．しかし，これらタスクに関する情報は実際に実行を行い得られたものではなく，予測値であるためこれらの情報を用いて最適化を行っても，実際の処理を行った際に最適な割当てになっているとは限らない．そこで，割当て時に与えられるタスク情報の予測値と実際の値が異なっていても，処理効率の低下が少ない割当てを行う必要がある．また，初期割当ては，処理の開始以前に行われるため，実行時のオーバヘッドとはならないが，割当てに要する時間は並列処理システムのオーバヘッドとなるため，より短い時間で割当てを行う必要がある．

（2）動的割当て

動的割当ては処理の実行時に行われる割当てである．動的割当てでは，ジョブの実行開始以前には，全くタスクを割当てずに，実行時にプロセッサが空き状態となれば，タスクを割当てる方式である．したがって，実行時に空き状態となるプロセッサが発生しにくいという特徴がある．また，初期割当てに対して，タスク情報はより正確なものとなるため，比較的良い割当て結果を得ることができる．しかしながら，処理実行時に個々のタスクごとに割当てを行い，また，タスクの割当てごとに，プロセッサ間でタスク情報やプロセッサの情報の交換を行わなければならないため，実行時のオーバヘッドが大きくなる．

（3）適応型割当て

適応型割当ても動的割当てと同様に，処理の実行時に行われる割当てであるが，動的割当てと異なり，処理の開始以前に各タスクは何らかの方法で初期割当てされており，処理を行う過程で，各プロセッサの負荷やプロセッサ間の通信量などを実測し，これらの情報をもとにタスクのプロセッサへの再割当てを行い処理効率を改善するものである（図2-3）．このように，適応型割当てではシステムの状況に応じて割当てを行うために高い処理効率を得ることが可能である．適応型割当ては，処理実行時に行うため実行時のオーバヘッドとなるが，各タスクの実行ごとに割当てる必要がないため，動的割当てよりも小さいオーバヘッドとなる．適応型割当てでは，より短い時間で，より高い処理効率に収束することが要求されるが，ジョブの大きさにより，比較的小さいジョブでは処理時間が短いため，収束処理効率の高さよりも，処理効率の改善に

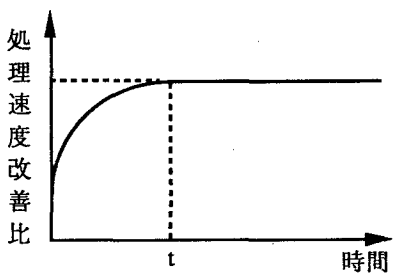


図2-3　適応型割当て

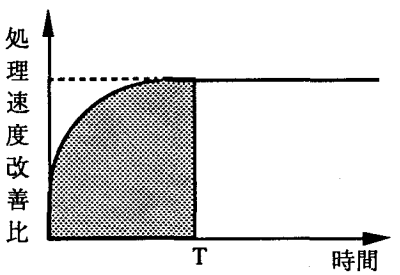


図2-4　適応型割当ての性能

要する時間が短い方がよく．また，比較的大きいジョブでは，処理時間が長いため，改善に要する時間の長さよりも，収束処理効率の高さが要求される．つまり，適応型割当てでは図2-4に示すように処理効率を処理開始時から終了時まで時間積分したものが大きい方がよい．

なお，本研究ではこれらの割当法のうち，初期割当法を新たに提案する．

2.3.3　プロセッサ間同期法

ここでは以下に示すような仮定を持つ並列処理システムを考える．

① 各プロセッサ共通の論理時刻を示す時計はなく，各プロセッサはそれぞれ独自の論理時刻を持つ．
② 各プロセッサは他のプロセッサに送信する際にそのときの論理時刻をメッセージに付加する．
③ メッセージを受信したプロセッサはメッセージによって要求されたタスクを実行する．

論理時刻とは，処理システムにおける抽象的な時刻であり，イベントの発生順序を規定するものである．論理時刻の値が小さいイベントは，より大きな値を持つイベントより前に発生する．たとえば図の4つのイベントでは［1］の論理時刻は［11］や［21］より小さい．また，［2］の論理時刻は［21］より小さい．一方，現実世界において物理的に流れている時刻を実時刻と呼ぶ．

(1) スケジュール可能なジョブの実行

図2-1に示したイベントの発生順序関係を持つようなジョブをプロセッサ1とプロセッサ2の2台のプロセッサからなる並列処理システムで実行する場合を考える．ここでは，イベント［1］，［11］に関するタスクがプロセッサ1に，イベント［2］，［21］に関するタスクがプロセッサ2に割当てられたとすると（図2-5），プロセッサ2では，まずイベント［2］が発生し終了する．そして，次のイベント［21］はプロセッサ1のイベント［11］が終了するまでは

発生できないことが予めわかっている．そこで，予めプロセッサ１では何らかの手段でプロセッサ２に対して［１１］の終了を伝えることにしておき，プロセッサ２では，［１１］の終了が伝えられるまでは［２１］の開始を禁止することでイベントの発生順序関係を正しく保つことができる．

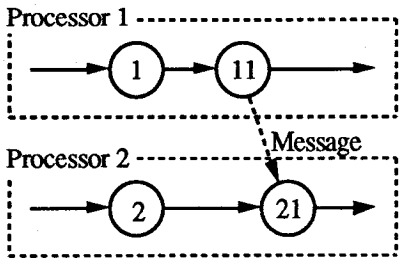


図2-5　スケジュール可能なジョブの実行

(2)スケジュール不可能なジョブの実行

次に図2-2に示したイベント発生順を持つジョブを２台のプロセッサからなる並列処理システムで実行する場合を考える．イベント［１］［１１］［１２］に対するタスクがプロセッサ１に，イベント［２］［２１］［２２］に対するタスクがプロセッサ２に割当てられたとすると（図2-6），イベント［１］と［１１］または［１２］の何れか

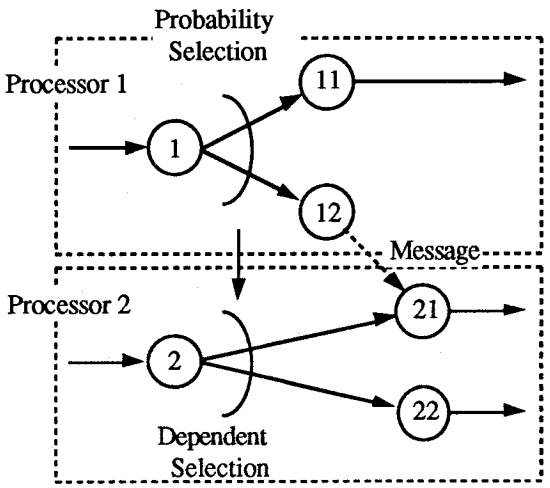


図2-6　スケジュール不可能なジョブの実行

一方の2つのイベントはプロセッサ1のイベント発生系列に含まれ，また［2］と［21］または［22］のいずれか一方はプロセッサ2のイベント発生系列に含まれる．

ここでプロセッサ2のイベント発生系列を眺めると，イベント［2］が終了した時点で，プロセッサ2はイベント［21］あるいは［22］を直ちに開始することはできない．なぜならば，イベント［21］はイベント［12］の終了以降に開始されなければならないからである．したがって，プロセッサ2はプロセッサ1から［12］の終了が伝えられるまでは［21］及び［22］の開始を保留しなければならない．しかも，もしプロセッサ1で［12］ではなく［11］が選ばれたとすると，プロセッサ2はいつまでもイベント［12］の終了が伝えられることがなく，そこで動作を停止する．これを回避する何らかの方策が必要となる．

このように，ジョブを並列処理システムで実行する際にはプロセッサ間で処理の進行の同期をとらなければ，タスクの発生順序関係に矛盾が生じたりデッドロックやロックアウトが生じてしまう．したがって，並列処理システムではプロセッサの処理の進行を制御するプロセッサ間同期法が必要となる．この，プロセッサ間同期法に必要な機能としては以下のようなものがある．

・同期ためのオーバヘッドが少ない．
・ジョブや並列処理システムの規模の増大によるオーバヘッドの増加が少ない．

また，プロセッサ間の同期法には集中制御方式と分散制御方式がある．集中制御方式は，他のプロセッサに対し処理の進行や停止を指示する制御プロセッサを設け，プロセッサの処理の進行を集中的に制御する方式である．集中制御方式では，全てのプロセッサの動作を集中的に管理するため，最適な同期を行うことが容易であるが，システムが大規模となると制御プロセッサの処理量や制御プロセッサと他のプロセッサとの通信が増加し，オーバヘッドが増加するという問題がある．また，プロセッサ台数の変更などのシステム構成の変更が容易に行えない．一方，分

散制御方式は，各プロセッサが各々独立に処理の進行・停止を決定し同期をとる方式である．この方式では，各プロセッサは自分自身の処理を最適制御しようとするが，システム全体で最適な制御が行われるとは限らない．しかしながら，各プロセッサの同期アルゴリズムに変更を加えることなく容易にシステム構成の変更を行うことができる．

本研究では，分散制御方式の1つである先行制御方式を対象に理論解析を行う．

2.4　結言

本章では，計算機システムで処理されるジョブを処理の開始以前に処理過程が決定されているスケジュール可能なジョブと，処理の実行にともなって処理過程が決定されるスケジュール不可能なジョブに分類した．

次に，ジョブを並列処理する際に必要となる実行制御方式である，ジョブ分割法，タスク割当法，プロセッサ間同期法について述べた．

ジョブ分割法は，並列処理システムで処理されるジョブを並列処理可能なタスクに分割する役割を担っており，より短い分割時間でジョブのもつ並列性を最大限に引き出すことが要求される．また，タスク割当てやプロセッサ間の同期をとる際に必要となるタスク情報の抽出を行う必要がある．本論文では，第3章でジョブ分割法を新たに提案する．

また，タスク割当法は，ジョブ分割により得られたタスクを，処理終了時間が短くなるようにプロセッサに割当てるものである，タスク割当法は，割当ての行われる時点により，処理の開始前に行われる初期割当て，処理の実行にともなって行う動的割当て，処理状況に応じて再割当てを行う適応型割当ての3種類に分類できる．本論文では，第4章で初期割当法を新たに提案する．

スケジュール不可能なジョブを実行する際には，プロセッサ間で処理の進行の制御を行わなければ，処理結果に矛盾が生じる．このプロセッサの処理の進行を制御するものがプロセッサ間同期法である．プロセッサ間同期法には，制御プロセッサを設け他のプロセッサの動作を集中的に管理する集中制御方式と各プロセッサが独立に処理の進行・停止を決

定して同期をとる分散制御方式がある．本論文では拡張性の点で優れている分散制御方式の1つである先行制御方式の理論解析を第5章で行う．

第3章

メッセージ依存分割法

3.1　緒言

現在実用化されている並列処理システムの多くは，元来複数でしかも独立しているようなジョブを複数のプロセッサに割当てることで高速化を実現している．しかしながら，単一ジョブを分割してプロセッサに割当てて実行することによって処理時間の短縮を可能としている並列処理システムは研究段階にある．

このような並列処理システムにおいては，ジョブをタスクと呼ばれる相互に並列実行可能な処理単位に分割するジョブ分割法が必要となる．ジョブ分割法を用れば，見かけ上並列性の現れていないプログラムであっても，適切な分割を施すことによって並列性を抽出することができ，得られたタスクを複数のプロセッサに割当てれば高速な処理が可能となる．また逐次処理言語によって記述された膨大なプログラムの蓄積を活用するためにも，こうしたジョブ分割法は有効であり重要である．

そこで本章では，まずタスクの並列実行性とタスク間の処理依存関係について考察する．次にジョブを記述したプログラム中でメッセージの送受信点を検出し，ジョブの分割を行うメッセージ依存分割法[MURA1 89][MURA1 90]を提案する．さらに，タスクの特性として特に処理依存関係に着目しこれを検出するアルゴリズムについて述べる．また，従来の分割法と比較し，メッセージ依存分割法が並列性抽出の点で優れていることを示す．

3.2　プログラム構成単位間の依存関係

図3-1に示すようなタスクを並列実行の処理単位とする並列計算のモデルを考える．タスクを各々自立した処理単位とみなし，各タスクはそれぞれ内部手続きと保持すべき内部状態を持つものとする．これらのタスクは，完全に並列実行が可能であることはなく処理の順序に関して制約を持つためタスク全体は処理順序に関して半順序集合をなす．この順序関係を定めるのはタスク間の処理依存関係である．処理依存に関する情報は，タスク間でのメッセージ通信という形で交換される．全体としてのジョブの実行は，ジョブ分割により生成されたこれらタスク間でのメッセージ通信によるタスクの起動やデータの受渡しと，個々のタスクの内部手続きの実行によって構成される．

この並列計算モデルは，たとえばこれらタスクをローカルメモリを持つプロセッサ群に割当て，メッセージ通信をプロセッサ間通信に対応させることによって実現できる．

ここでタスク間の処理依存関係について更に分類を行うと，処理依存関係には制御依存関係とデータ依存関係の2種類が存在する．

制御依存関係は，先行するタスクの処理結果に応じて後続のタスクの処理開始が決定される場合に生じる．条件分岐や関数呼出しは，制御依存関係である．制御依存関係の例を図3-2に示す．

データ依存関係は共有変数の書込み・参照関係によって発生し，処理順序を規定する．データ依存関係の例を図3-3に示す．

このデータ依存関係は，図3-4に示した従来のジョブ分割法におけるデータ依存関係の概念では図3-4（a）のフロー依存関係に相当する．しかしながら，フロー依存関係以外の逆依存関係及び出力依存関係については，処理順序を規定する関係とはならない．なぜならば，この並列計算モデルにおいては，逆依存関係及び出力依存関係の存在するタスク間ではメッセージの送信は起こらず，相互に並列実行が可能となるからである．

ここでタスクとメッセージの並列計算モデルに基づいたジョブ分割による並列性の抽出と処理依存関係の検出の例を図3-5に示す．図3-5（a）のソースプログラムを分割し，図3-5（b）のタスクを生成したとする．ここでT1とT2，T3とT4はそれぞれ共有変数によって

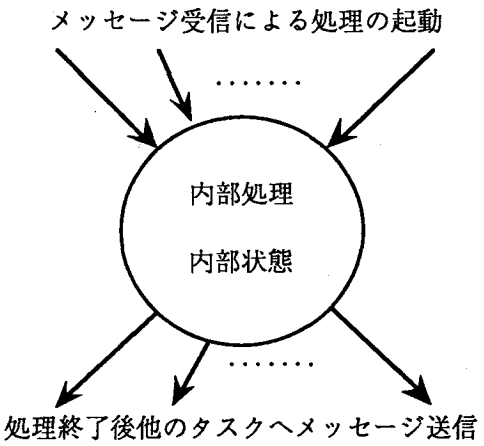


図3-1　タスクのモデル

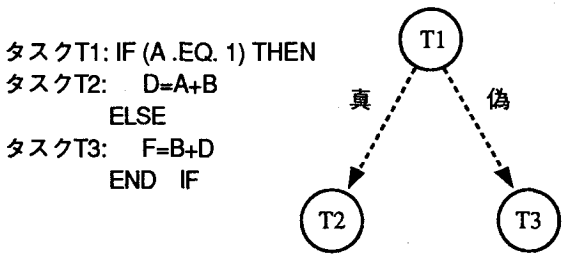


図3-2　制御依存関係

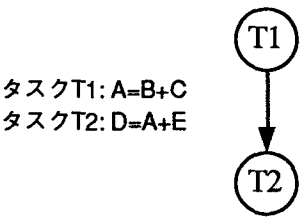
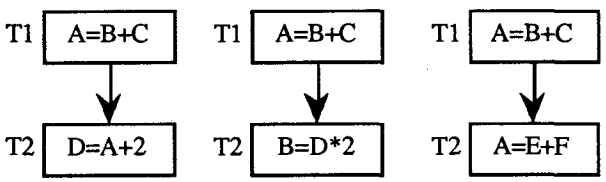


図3-3　データ依存関係



(a) フロー依存　(b) 逆依存　(c) 出力依存

図3-4　従来のデータ依存関係の概念

データ依存関係にある．また従来のデータ依存の概念によればＴ２とＴ３の間は逆依存関係にあることがわかる．これらのデータ依存関係の制約から，逐次処理計算機で実行した場合には図３-５（ｃ）のような実行の流れになる．一方，並列計算を行った場合には図３-５（ｄ）に示すように，Ｔ１とＴ２，Ｔ３とＴ４に関してはそれぞれデータ依存関係により処理順序の制約が存在するが，Ｔ１－Ｔ２とＴ３－Ｔ４は互いに並列実行が可能となる．これは各タスクが内部に変数の値を保持し内部の処理を終えた後データを他のタスクに送信する機能をもつために，Ｔ２とＴ３の逆依存関係による処理順序が不必要となるためである．すなわちタスクとメッセージの並列計算モデルに基づいたジョブ分割法によって，逐次計算における共有変数の操作順序から発生する処理依存関係が取り除かれ，潜在的な並列実行性が引き出されている．

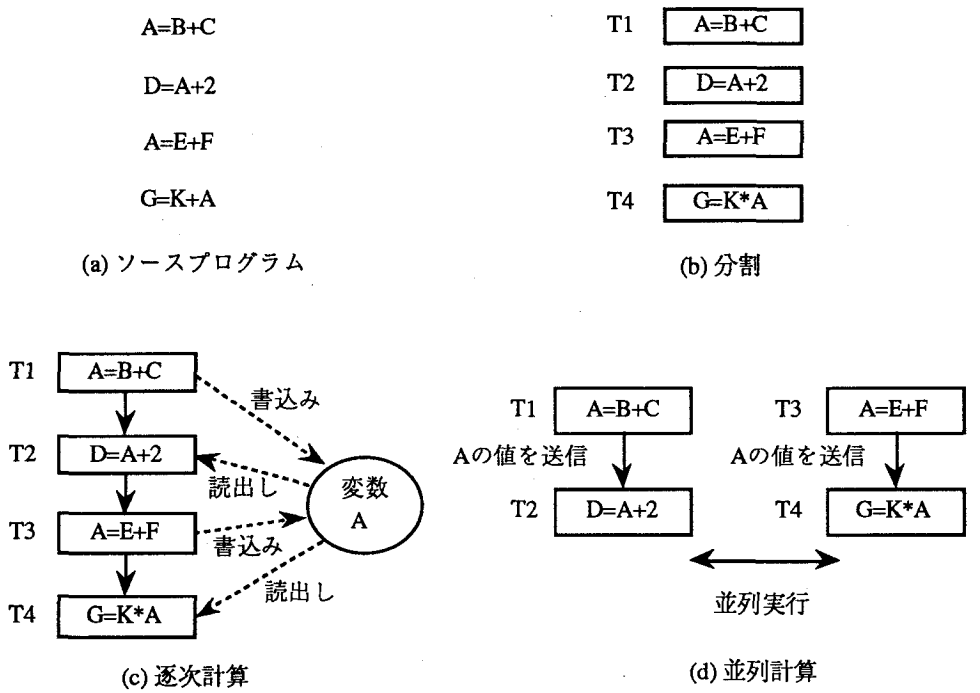


図３-５　ジョブ分割による並列性の抽出

このようにして，ジョブの持つ並列性を抽出し，同時に並列実行に必要な処理依存関係を検出することがジョブ分割法の重要な役割である．

図3-1の並列計算モデルに従うようなタスクとメッセージを，実際のプログラムから生成するために，ここで改めてタスクの定義を行う．またジョブ分割法がどの程度効率よく並列性を抽出しているかを判断する基準として理想的な分割の定義も行う．

（定義）
ジョブJ ≡ {処理依存関係による半順序関係を持った
処理要素の集まり}

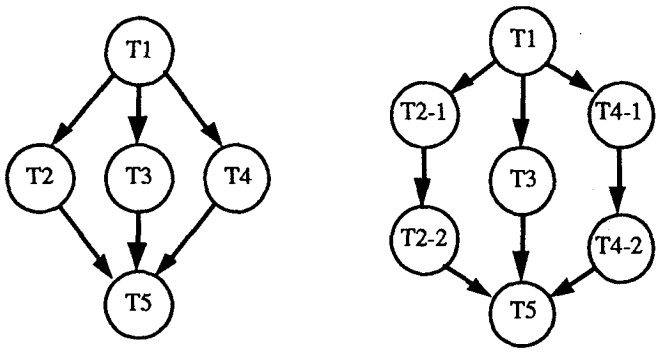
タスク集合T ≡ {ジョブの部分集合Tn．
ただしn≠mに対してTn∩Tm＝φ，∪Tn＝J}

（定義）
あるタスク集合T＝ {Tn} において，任意のタスク間で1対1の通信を行わず，またタスク集合Tの要素数が最大となるとき，このタスク集合Tを得るような分割を理想的な分割という．

理想的な分割は，ジョブのタスクへの分割回数を最小に抑えながら最大の並列性をジョブから抽出する．

以上の定義の下では，理想的な分割によって得られるタスク集合において開始タスク及び終了タスクを除いた全てのタスクは，内部の処理の開始時に他のタスクよりメッセージを受信し，処理中にはメッセージの送受信を行わず，処理の終了時に他のタスクへメッセージを送信する．

理想的な分割と理想的でない分割の違いを示す例を図3-6にあげる．図3-6（a）ではT1－T2－T5，T1－T3－T5，T1－T4－T5の3つの互いに並列実行可能な系列を抽出していることがわかる．一方，図3-6（b）においてもやはり3つの並列実行可能な系列を抽出しており，並列性の抽出という観点からはどちらの分割も同等と言える．しかしながら図3-6（a）におけるT2及びT4は図3-6（b）では更に分割されている．これらの分割は並列性の抽出には寄与せず，しかも並列処理システムでの実行においてはオーバヘッドにつながる．



(a) 理想的な分割　　(b) 理想的でない分割

図3-6　理想的な分割

このことから，図3-6（a）の分割は並列性を十分抽出し，しかも分割によるオーバヘッドの少ない理想的な分割であることが言える．

3.3　メッセージ依存分割法の提案

ジョブを記述するプログラムを分割する際，メッセージ通信の発生する可能性の高い位置で分割を行えば，より理想的な分割に近いジョブ分割が実現できる．このようなタスク間のメッセージに着目した分割法として，メッセージ依存分割法を提案する．

このメッセージ依存分割法は，プログラム中のメッセージを明らかにすることによって分割を行うため，特定の言語には依存しない．しかしジョブがCやＦＯＲＴＲＡＮ等の言語で記述されている場合，全てのメッセージの送受信を明らかにすることは容易ではない．したがって必要な分割が行われない，あるいは無駄な分割が行われるといったことが起こりうる．

以下に分割アルゴリズムと処理依存関係の検出法について述べる．

・分割アルゴリズム
メッセージ依存分割法は，以下のアルゴリズムでジョブ分割を行う．

（ステップ1）ジョブを記述したプログラム中で，メッセージの送受信点を全て見いだす．
（ステップ2）見いだしたメッセージの送受信点の直後を分割位置として前後に分割する．

ここでジョブの記述言語として，特殊なライブラリルーチンを一切用いないＦＯＲＴＲＡＮ７７を仮定する．このとき次のような記述がメッセージとなる可能性が高いと考えられ，実際の分割位置となる．

分割位置：プログラムの流れを制御する実行文を含む行．ＣＡＬＬ，ＩＦ（条件式）ＴＨＥＮ，ＥＬＳＥ，ＥＮＤ　ＩＦ，ＤＯ，ＣＯＮＴＩＮＵＥ，ＥＮＤ

またＦＯＲＴＲＡＮ特有の記述に対応して定める細則は次のとおりである．

（細則1）ＤＯ文による繰り返し処理
繰り返し処理単位の末尾はＣＯＮＴＩＮＵＥ文となっているものと仮定し，繰り返し処理の先頭のＤＯ文の直後の行から末尾のＣＯＮＴＩＮＵＥ文までの実行文の集合をＤＯループと呼ぶ．ＤＯループ内部に分割点が存在しない場合はＤＯループ全体を一つのタスクとする．分割点が存在する場合には，ループの繰り返し回数がプログラムの実行開始以前に定まっている場合に限り，繰り返しにおいて異なる要素全てに対してタスクを一つずつ生成する．

（細則2）異なるＣＡＬＬ文による同一サブルーチンの呼び出し
プログラム中で異なった位置にあるＣＡＬＬ文において同一サブルーチンを呼び出している場合は各ＣＡＬＬ文に対応したタスクがそれぞれ別個に生成されるものとする．

・処理依存関係の検出
ジョブ分割により生成されるタスクの集合は，並列処理システムにお

ける次の処理段階であるプロセッサへのタスク割当て及び複数プロセッサでの実行へと引き渡されていく．そしてジョブの実行段階においては各プロセッサに割当てられたタスクは相互の処理依存関係にしたがって必要な情報をタスク間通信によって交換する．このとき，異なるプロセッサに割当てられたタスク間のメッセージによる通信はプロセッサ間通信を引き起こし，並列処理におけるオーバーヘッドとなる．したがってタスクの割当て・同期実行を行うために個々のタスクの持つ処理量やタスク相互の処理依存関係といったタスクの持つ特性を，並列実行性という観点から検出する必要がある．

ここではタスク間の処理依存関係をタスク特性として着目し，その検出アルゴリズムを示す．

まず，ジョブ分割によって得られたタスク間の処理依存関係は，制御依存関係とデータ依存関係の2種類に分類できる．

制御依存関係は条件分岐や手続き呼出によって発生する．ＦＯＲＴＲＡＮの場合にはブロックＩＦ文，ＣＡＬＬ文，ＤＯ文において分割・生成されたタスク間には制御依存関係が存在する．

一方，データ依存関係は次のアルゴリズムによって検出される．

（ステップ1）分割・生成されたタスクに対し，逐次処理した場合の処理順序に従った番号付けを行う．これを処理順序番号と定義し，タスクＴの処理順序番号をｎｕｍ（Ｔ）と表記する．

（ステップ2）タスクＴにおいて参照している全ての変数の集合をＶｒ（Ｔ）とする．各変数ｖ∈Ｖｒ（Ｔ）について，ｖの書込みを行い，かつｎｕｍ（Ｔｗ（ｖ））＜ｎｕｍ（Ｔ）であるようなタスク集合｛Ｔｗ（ｖ）｝を求める．

（ステップ3）｛Ｔｗ（ｖ）｝の要素の中で最も処理順序番号の大きなタスクＴｄを求める．Ｔｄが存在するとき，ＴｄとＴはデータ依存関係にある．各変数ｖ∈Ｖｒ（Ｔ）に対してＴｄを求める．

（ステップ4）任意のタスクＴについて（ステップ2）～（ステップ3）を行う．

タスク集合に対してこのアルゴリズムを適用することにより，データ依存関係を全て検出することができる．

図3-7を例にデータ依存検出アルゴリズムの適用を説明する．

（1）タスクT1～T4の処理順序番号が図のように求められるものとする．

（2）たとえばT4についてデータ依存関係を求めると次のようになる．T4において参照している変数はAであり，Aの書込みを行っているタスクであって処理順序番号がT4より小さいものはT1及びT3である．

（3）T1とT3で処理順序番号が大きいものはT3である．よってT3とT4はデータ依存関係にある．

（4）T1，T2についても同様にしてデータ依存関係が求められる．

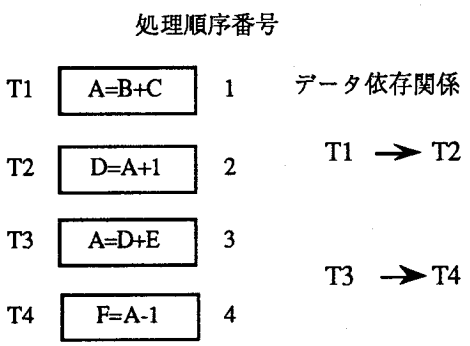


図3-7　データ依存関係の検出例

3.4　メッセージ依存分割法の評価

比較対象として，ベクトル化[PADU 86]及びモジュール分割法[KIKU 88]を選択した．これらの分割法とメッセージ依存分割法とを同一のプログラムに適用し，分割回数と並列性抽出度を評価尺度に用いて比較した．

ここでベクトル化及びモジュール分割法の基本的なアルゴリズムとその特徴について簡単にまとめる．

ベクトル化

ベクトル演算など同時処理可能な計算をベクトルプロセッサで並列に処理するために，プログラムのループ部分を各ループ要素毎に分割する．

- 分割アルゴリズムが簡単であり，ベクトルプロセッサというハードウェアに適合した並列性が抽出可能である．
- 繰り返しの各要素に対応した処理が互いに依存する実行順序を持つ場合には分割を行っても並列性は得られない．
- ＤＯループというプログラムの一部分に注目して並列化するため，必ずしもプログラム全体の並列処理を考えた場合の最適な分割が行えない．

モジュール分割法

ＦＯＲＴＲＡＮなどの逐次処理型高水準言語による大規模なプログラムを対象とする．プログラムはモジュールと呼ばれる処理単位で構成されているとみなし，このモジュールに着目して分割を行う．

- プログラム記述全体にわたって分割を行うため，ＤＯループにとどまらない広い範囲からの並列性を抽出できる．
- モジュールに対して再帰的に分割を行い，また隣接文で互いに依存関係がなければ分割を行うため，分割回数が増し数多くのタスクが生成される．

以上２種類の分割法とメッセージ依存分割法を図３-８に示す統計処理を行うプログラムに対して適用し，それぞれの場合についてタスクフローグラフと並列性抽出度を求めて各分割法の比較・検討を行う．

```
 1:      PROGRAM SOUKANKEISU
 2:        REAL X(10),Y(10),MEANX,MEANY
 3:        REAL VARX,VARY,COVXY,RXY
 4:        READ(5,*)X,Y
 5:        CALL MEAN(X,MEANX)
 6:        CALL MEAN(Y,MEANX)
 7:        CALL VAR(X,MEANX,VARX)
 8:        CALL VAR(Y,MEANY,VARY)
 9:        CALL COV(X,MEANX,Y,MEANY,COVXY)
10:       RXY=COVXY/(SQRT(VARX)*SQRT(VARY))
11:        WRITE(6,*)RXY
12:      END
13:
14:      SUBROUTINE MEAN(DATA,M)
15:        REAL DATA(10),M
16:        M=0
17:        DO 10 I=1,10
18:          M=M+DATA(I)
19: 10     CONTINUE
20:        M=M/10
21:             WRITE(6,*)M
22:      END
23:
24:      SUBROUTINE VAR(DATA,M,V)
25:        REAL DATA(10),M,V
26:        V=0
27:        DO 20 I=1,10
28:          V=V+(DATA(I)-M)**2
29: 20     CONTINUE
30:        V=V/(10-1)
31:             WRITE(6,*)V
32:      END
33:
34:      SUBROUTINE COV(DATA1,M1,DATA2,M2,C)
35:        REAL DATA1(10),M1,DATA2(10),M2,C
36:        C=0
37:        DO 30 I=1,10
38:         C=C+(DATA1(I)-M1)*(DATA2(I)-M2)
39: 30     CONTINUE
40:        C=C/(10-1)
41:      END
```

図3-8　相関係数を求めるプログラム

このプログラムは2つの配列データの相関係数を計算するものである．まず要素数が１０個の配列ＸとＹに各々１０個ずつ数値を読み込み，各配列に格納されたデータの平均値・分散・共分散をサブルーチンによって計算した後，相関係数を求め出力する．なお，リスト左端の行番号は説明のために付与した．

（ａ）メッセージ依存分割法を適用した場合

行番号５，６，７，８，９において各々平均，分散及び共分散を計算するサブルーチンを起動するメッセージを送信しており，この位置で分割される．各サブルーチン内部は繰り返し処理の部分全体が１つのタスクとして分割される．分割によって得られるタスクを表３-１に示す．また，タスクフローグラフを図３-９に，各パス長を表３-２に示す．なお，表３-１におけるサイズはジョブ全体の処理量に対する各タスク個々の処理量の占める割合を表す．

（ｂ）モジュール分割法を適用した場合

平均，分散，共分散を求める手続きが分割され，更に各サブルーチン内部のＤＯループも繰り返し回数に等しい個数のタスクに分割される．分割によって得られるタスクを表３-３に示す．またタスクフローグラフを図３-１０に，各パス長を表３-４に示す．

（ｃ）ＤＯループのベクトル化を適用した場合

このプログラム例に対しては各サブルーチン内部のＤＯループが繰り返し回数に等しい個数のタスクに分割される．分割によって得られるタスクを表３-５に示す．またタスクフローグラフを図３-１１に，各パス長を表３-６に示す．

以上の結果より，各分割法のタスク数，パスの本数，並列性抽出度を表３-７に示す．タスク数は，メッセージ依存分割法が１７個であるのに対し，モジュール分割法は６２個，ベクトル化では５６個と大きく異なっている．これはメッセージ依存分割法がＤＯループに対しては分割を行わないのに対しモジュール分割法とベクトル化の２つはＤＯループを各要素毎に分割するためである．

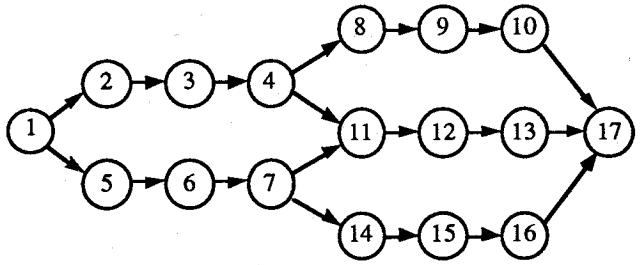


図3-9　メッセージ依存分割法による
図3-8のタスクフローグラフ

表3-1　メッセージ依存分割法によるタスク

ﾀｽｸNo	行番号	処理内容	Size
T1	1～4	データの入力	1/63=0.016
T2	16	X平均・変数の初期化	1/63=0.016
T3	17～19	〃　要素の和	10/63=0.16
T4	20	〃　平均の計算	1/63=0.016
T5	16	Y平均・変数の初期化	1/63=0.016
T6	17～19	〃　要素の和	10/63=0.16
T7	20	〃　平均の計算	1/63=0.016
T8	26	X分散・変数の初期化	1/63=0.016
T9	27～29	〃　要素の自乗和	10/63=0.16
T10	30	〃　分散の計算	1/63=0.016
T11	26	Y分散・変数の初期化	1/63=0.016
T12	27～29	〃　要素の自乗和	10/63=0.16
T13	30	〃　分散の計算	1/63=0.016
T14	36	共分散・変数の初期化	1/63=0.016
T15	37～39	〃　繰り返し処理	10/63=0.16
T16	40	〃　共分散の計算	1/63=0.016
T17	11	相関係数の計算，出力	2/63=0.032

表3-2　メッセージ依存分割法によるＰａｔｈ

Path	タスクフロー	Pathの値
Path1	T1-T2-T3-T4-T8 -T9 -T10-T17	27/63=0.43
Path2	T1-T2-T3-T4-T14-T15-T16-T17	27/63=0.43
Path3	T1-T5-T6-T7-T11-T12-T13-T17	27/63=0.43
Path4	T1-T5-T6-T7-T14-T15-T16-T17	27/63=0.43

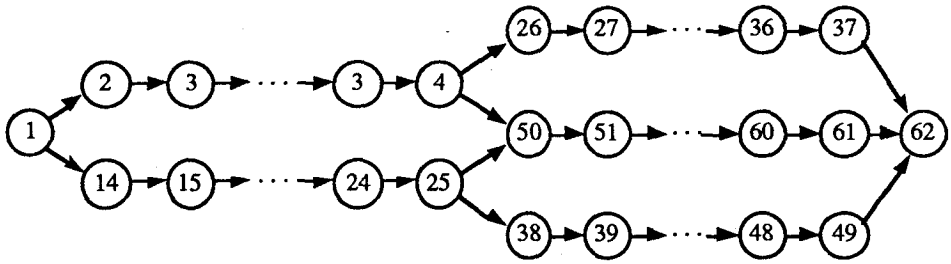


図3-10　モジュール依存分割法による
図3-8のタスクフローグラフ

表3-3 モジュール分割法によるタスク

ﾀｽｸNo	行番号	処理内容	Size
T1	1～6	データの入力	1/63=0.016
T2	16	X平均・変数の初期化	1/63=0.016
T3～T12	17～19	〃　要素の和	1/63=0.016
T13	20	〃　平均の計算	1/63=0.016
T14	16	Y平均・変数の初期化	1/63=0.016
T15～T24	17～19	〃　要素の和	1/63=0.016
T25	20	〃　平均の計算	1/63=0.016
T26	26	X分散・変数の初期化	1/63=0.016
T27～T36	27～29	〃　要素の自乗和	1/63=0.016
T37	30	〃　分散の計算	1/63=0.016
T38	26	Y分散・変数の初期化	1/63=0.016
T39～T48	27～29	〃　要素の自乗和	1/63=0.016
T49	30	〃　分散の計算	1/63=0.016
T50	36	共分散・変数の初期化	1/63=0.016
T51～T60	37～39	〃　繰り返し処理	1/63=0.016
T61	40	〃　共分散の計算	1/63=0.016
T62	11	相関係数の計算，出力	2/63=0.032

表3-4　モジュール分割法によるＰａｔｈ

Path	タスクフロー	Pathの値
Path1	T1-T2-T3-･･･-T12-T13- -T26-T27-･･･-T36-T38-T62	27/63=0.43
Path2	T1-T2-T3-･･･-T12-T13- -T50-T51-･･･-T60-T61-T62	27/63=0.43
Path3	T1-T14-T15-･･･-T24-T25- -T38-T39-･･･-T48-T49-T62	27/63=0.43
Path4	T1-T14-T15-･･･-T24-T25- -T50-T51-･･･-T60-T61-T62	27/63=0.43

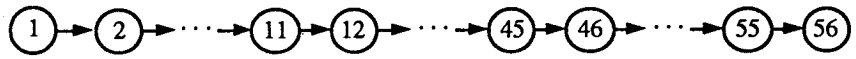


図3-11　ベクトル化による図3-8の
タスクフローグラフ

表3-5　ベクトル化によるタスク

ﾀｽｸNo	行番号	処理内容	Size
T1	1～6，16	データの入力及びX平均・変数の初期化	2/63=0.032
T2～T11	17～19	X平均・要素の和	1/63=0.016
T12	20，16	X平均・平均の計算，Y平均・変数の初期化	2/63=0.032
T13～T22	17～19	Y平均・要素の和	1/63=0.016
T23	20，26	Y平均・平均の計算，X分散・変数の初期化	2/63=0.032
T24～T33	27～29	X分散・要素の自乗和	1/63=0.016
T34	30，26	X分散・分散の計算，Y分散・変数の初期化	2/63=0.032
T35～T44	27～29	Y分散・要素の自乗和	1/63=0.016
T45	30，36	Y分散・分散の計算，共分散・変数の初期化	2/63=0.032
T46～T55	37～39	共分散・繰り返し処理	1/63=0.016
T56	40，11	共分散・共分散の計算，相関係数の計算，出力	3/63=0.048

表3-6　ベクトル化によるＰａｔｈ

Path	タスクフロー	Pathの値
Path1	T1-T2-･･･-T11-T12-T13-･･･-T22-T23-T24-･･･-T33-T34-T35-･･･-T44-T45-T46-･･･-T55-T56	63/63=1.0

表3-7　図3-8の分割結果の比較

	メッセージ依存分割法	モジュール分割法	ベクトル化
タスク数	17	62	56
Ｐａｔｈ本数	4	4	1
並列性抽出度	2.3	2.3	1.0

またメッセージ依存分割法とモジュール分割法がともに並列性抽出度Ｃ＝２．３であるのに対し，ベクトル化では並列性抽出度Ｃ＝１．０となり，全く並列性が抽出されていない．これは，プログラム例の繰り返し処理の部分はループの１回毎の演算が前回の演算結果に依存しており分割しても並列実行ができないにもかかわらず，ベクトル化ではＤＯループ部分に対してのみ分割を施すためである．

また，モジュール分割法では，手続き呼び出しを分割してメッセージ依存分割法と同程度の並列性を抽出している．しかしながらモジュール分割法ではＤＯループも分割し，タスク数がメッセージ依存分割法の１７に比べ６２と多い．しかし，この分割は並列性の抽出には寄与しない．したがってこの分割対象例に関してはモジュール分割法はメッセージ依存分割法に比べて，無駄な分割を行っていると結論づけられる．

以上の比較から対象例として用いたジョブのようにプログラム自体が階層的構造を持ち，なおかつ繰り返し処理の内部が並列実行性を持たないようなジョブに対しては，メッセージ依存分割法は他の分割法に比べて少ない分割回数で十分な並列性を抽出できることが示された．

しかしながら，ジョブが階層的構造を持たない，あるいはループ内部の処理を並列実行できる繰り返し処理がジョブ全体に対して大きな割合を占めている場合には，メッセージ依存分割法のみの適用では十分な並列性が得られない．このような場合に対してはベクトル化やモジュール分割法のようなＤＯループの分割を併用する必要がある．

次に図３-１２に示す名簿の検索を行うプログラムに対して分割を行う．このプログラムで記述されるジョブはスケジュール不可能なジョブである．メッセージ依存分割法はスケジュール不可能なジョブに対して

も分割が行えるが，モジュール分割法，ベクトル化では分割できない．そこで，メッセージ依存分割法による分割のみ行う．

このプログラムでは，２つの一次元配列構造のデータに対して検索を行う．一つは名前と住所が１００人分登録されており，もう一つは名前と電話番号である．データの並び方は，名前の辞書順である．検索を開始するに当って１から３のいずれかの番号が入力され更に検索のキーとなる名前が入力される．１のときは電話番号，２のときは住所，３のときは電話番号と住所の両方の検索をする．また検索方法は２通りあり，キーの最初の文字がａ～ｍのときは，配列の先頭から検索する．またｎ～ｚのときは配列の末尾から先頭に向かって検索する．

ジョブは表3-8に示すタスクに分割される．各タスクのサイズについては次のようにして求められる．

一つの配列の検索のための処理量の期待値は２５になるものと考えられる．したがって，逐次処理した場合の平均ステップ数は，電話番号のみ検索の場合と住所のみ検索の場合と両方検索の場合を考え，更に入力処理と条件判断処理を考慮して求められる．すなわち

（逐次平均処理量）

＝　（４＋１＋１＋２５＋２）×１／３
　＋（４＋１＋１＋１＋２５＋２）×１／３
　＋（４＋１＋１＋１＋１＋２５＋２＋２５＋２）×１／３
＝１２９／３
＝４３

となる．この値を用いて各タスクの処理量を正規化することにより求めたサイズを，表3-8に示す．

次にタスク間の確率的なメッセージ通信を考慮したタスクフローグラフを図3-13に示す．

図3-13のタスクフローグラフでは，タスク５または６を起動する場合のタスクの処理系列においては等しい長さのパスが１００本存在する．一方，タスク７及び８を起動するような処理系列においては等しい長さのパスが合計２００本存在する．いずれの場合もパスの長さは等しく，

```
1: #include <stdio.h>
2: typedef struct{
3:          char *name;
4:          char *data;
5: } DATA;
6: static DATA telNumList[] = {
                                    "nameA",    "03-xxx-xxxx",
                                              ...
                                    "nameX",    "06-xxx-xxxx"
7:                                 };
8: static DATA adrsList[] = {
                                  "nameA",    "TOKYO...",
                                            ...
                                  "nameX",    "OSAKA..."
9:                                };
10:
11: main()
12: {
13:          int select;
15:          char key[32];
14:
16:          printf("Select Number?¥n");
17:          scanf("%d", &select);
18:          printf("Input Key Name?¥n");
19:          scanf("%s", key);
20:
21:          if(select == 1)
22:                    lookUpAndAnswer(telNumList, key);
23:          if(select == 2)
24:                    lookUpAndAnswer(adrsList, key);
25:          if(select == 3){
26:                    lookUpAndAnswer(telNumList, key);
27:                    lookUpAndAnswer(adrsList, key);
28:           }
29: }
30:
31: lookUpAndAnswer(DATA list[], char key[])
32: {
33:          int i;
34:          char keytop = *key;
35:
36:          if(keytop <= 'n'){
37:                    for(i = 0;i < 100;i++)
38:                              if(strcmp(key, list[i].name) == 0){
39:                                        printf("%s¥n", list[i].data);
40:                                         break;
41:                               }
```

```
42:         }else{
43:                 for(i = 99;i >= 0;i--)
44:                         if(strcmp(key, list[i].name) == 0){
45:                                 printf("%s¥n", list[i].data);
46:                                  break;
47:                          }
48:         }
49: }
```

図３-１２　データ検索のプログラム

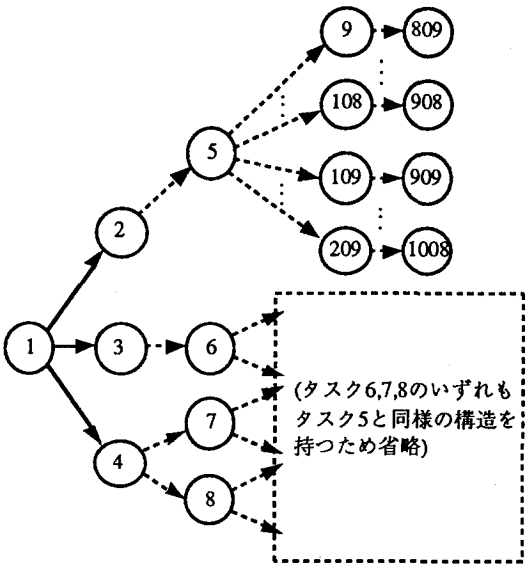


図３-１３　メッセージ依存分割法による
図３-１２のタスクフローグラフ

表３-８　メッセージ依存分割法によるタスク

ﾀｽｸNo	行番号	処理の内容	Size
T1	16-19	検索対象データ・キーの入力	4/43
T2	21	電話番号表の選択	1/43
T3	23	住所表の選択	1/43
T4	25	電話番号表及び住所表の選択	1/43
T5～T8	36	検索方向の選択	1/43

$$
\begin{aligned}
Path = &\ (4+1+1+1+2) \times 1/2 \\
&+ (4+1+1+1+2) \times 1/2 \\
= &\ 9
\end{aligned}
$$

となる．したがって，並列性抽出度Cは，

$$C = 43/9 \fallingdotseq 4.8$$

となる．

メッセージ依存分割法と従来の分割法を，いくつかのプログラムに適用し比較検討を行い，以下のような結果が得られた．

1）メッセージ依存分割法は他の分割法に比べて分割回数を低く抑え，高い並列性抽出度を得ることができる．

2）従来の分割法がスケジュール不可能なジョブを分割できないのに対し，メッセージ依存分割法はスケジュール不可能なジョブも分割可能である．

3.5　結言

本章では，逐次プログラミング言語で記述されたジョブの分割を行うジョブ分割法として，メッセージ依存分割法を提案した．メッセージ依存分割法は，プログラム中のメッセージの送受信点を検出し，ジョブの分割を行うものである．本方式で得られるタスクは，オブジェクト指向のオブジェクトと同様にメッセージを受け取ると起動し，終了時には他のプロセッサに対してメッセージの送信を行う．

さらに，本章ではメッセージ依存分割法と従来の方法を比較し，従来のジョブ分割に比べて，より高い並列性を抽出でき，また同程度の並列性を抽出する際には分割回数が少ないことを示した．また，メッセージ依存分割法では，メッセージの送受信点にのみ注目して分割を行うため，従来の分割法では分割できなかったスケジュール不可能なジョブに対しても分割を行うことができることを示した．

第4章

タスク多重割当法

4.1　緒言

並列処理システムにおけるタスク割当法としてこれまでにスケジュール可能なジョブを対象としたいくつかの割当法が提案されている．しかしながら，多くの場合並列処理システムに与えられるジョブはスケジュール不可能なジョブであり，これらのジョブを分割して得られるタスク集合にはその発生や処理量が確率的に変動するようなタスクが含まれる．このようなタスク集合に対し従来の割当法を適用しても，処理時間の短縮化が図れる割当て結果を得ることはできない．

本章では，確率タスク間には，いかなる実行系列においても共に発生することのない排反関係があることに注目し，これらの互いに排反なタスクを同一プロセッサへ割当てる“タスク多重割当法”[KOBA190a][KOBA190b]を提案する．

まず，タスク多重割当法において重要な概念である“確率タスク間の排反関係”について述べる．次に，確率タスクの多重割当てについて述べると共に，タスク多重割当法での優先順位決定法，確率タスクに対する処理量の取り扱いについて述べる．また，タスク多重割当法のタスク割当てアルゴリズムについて詳説する．さらに，タスク多重割当法と排反性を考慮しないCP/MISF法とを例を通して比較検討する．

4.2　確率タスク間の排反関係

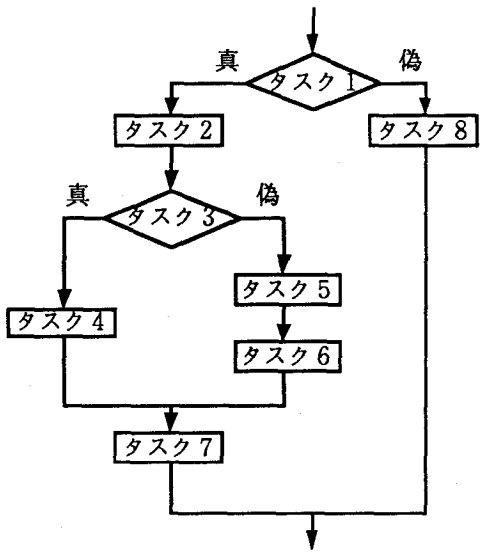
タスク集合の中にはいかなる実行系列においても共に発生することのないタスクの組がある．たとえば，図4-1に示すプログラムを例にとると，以下に示す2つの確率タスクの組は，いかなる実行系列においても2つのタスクが共に発生することがない．

タスク2とタスク8　タスク3とタスク8　タスク4とタスク5
タスク4とタスク6　タスク4とタスク8　タスク5とタスク8
タスク6とタスク8　タスク7とタスク8

このように，いかなる実行系列においても共に発生することがない確率タスク間の関係を“確率タスク間の排反関係”と呼び，排反関係にある確率タスクを“互いに排反な確率タスク”と呼ぶことにする．

```
IF（タスク1） THEN
    タスク2
    IF（タスク3） THEN
        タスク4
    ELSE
        タスク5
        タスク6
    END  IF
    タスク7
ELSE
    タスク8
END  IF
```

〈　リスト　〉



〈　フローチャート　〉

図4-1　条件分岐を含むプログラム

このような互いに排反なタスクは，タスク間の制御依存関係を用いて以下のように判断することができる．

(1) タスクTと同じ先行タスクを持ち，タスクTと発生する条件の異なるタスクは，タスクTと互いに排反なタスクである．
(2) 先行タスクがタスクTと互いに排反ならその後続タスクもタスクTと排反である．

たとえば，図4-1に示すような確率タスクを含むジョブをタスク間の制御依存関係のみに注目した場合，確率タスク間の関係は図4-2に示すような木構造となる．

このグラフにおいてタスク4と排反なタスクは，タスク4と同じく先行タスクがタスク3であり，その発生条件がタスク4とは異なるタスク5，タスク6，およびタスク4の先行タスクであるタスク3と互いに排反であるタスク8の3つである．

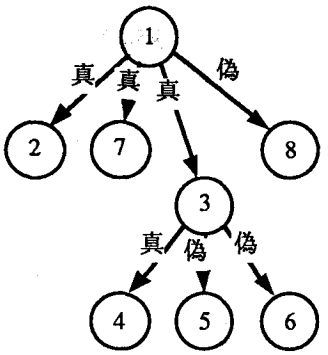


図4-2　制御依存関係

4．3　タスク多重割当法の提案

本論文で提案する“タスク多重割当法”は，CP法，CP/MISF法と同様にリストスケジューリング[ADAM 74][TOMI 89]の一種であるが，従来の方式と異なり確率タスク間の排反関係に着目し，互いに排反な確率タスクを同一プロセッサに割当てる“多重割当て”を行う．

この多重割当てを行うと，実際の処理系列においてプロセッサが何も処理を行わないアイドル状態の発生を抑制することができる．もし，多重割当てを行わずに確率タスクをプロセッサに割当てると，実際の処理においてその確率タスクが発生しなければ，プロセッサは空き状態となるが，タスク間の半順序関係を維持するために，他のタスクの実行を開

始できないことがある．しかしながら，多重割当てを行うと，多重割当てされたタスクの1つが発生しなくても，多重割当てされた他のタスクが発生すればプロセッサはアイドル状態とはならない．また，多重割当てされた互いに排反なタスクが1台のプロセッサを巡り競合することはない．なぜならば，互いに排反なタスクはいかなる実行系列においても共に発生することはない．したがって，たとえ同一のプロセッサに割当てられても，実際の実行の際には，どちらか一方のタスクの処理のみが行われ，もう一方のタスクの処理を行う必要がないからである．図4-3は排反関係を考慮した場合と考慮しない場合の割当て結果と実際の処理系列である．この図から分かるように，排反性を考慮し，互いに排反なタスクを同一のプロセッサに割当てた場合は，排反性を考慮しない割当てに比べタスク1の処理結果が真の場合も偽の場合も処理時間が短くなる．

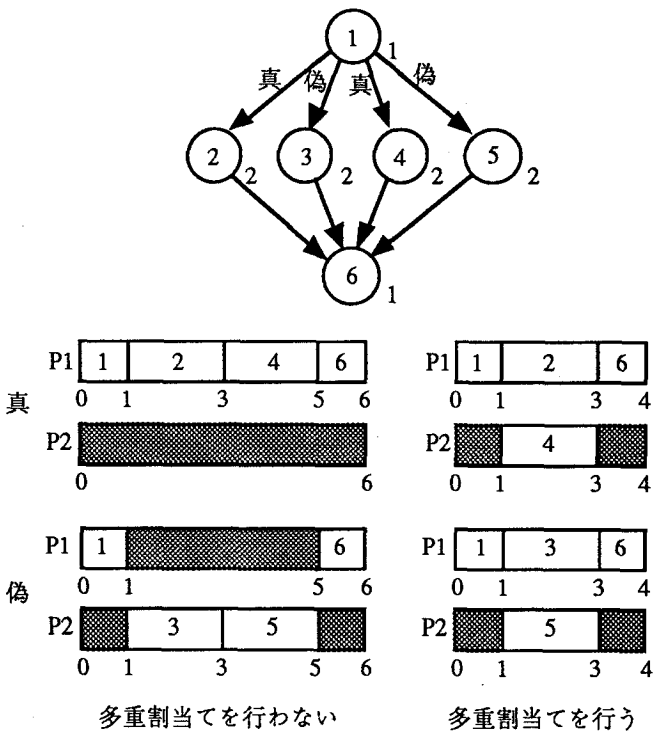


図4-3　多重割当て

（1）　確率タスクの処理量

本割当法では，確率タスクを含むタスク集合が与えられた場合，直後タスクから見た確率タスクTの処理量を，その直後タスクと確率タスクTとの制御依存関係により異なる値として取り扱う．

確率タスクTの直後タスクをタイプⅠ，Ⅱの2つに分類し，タイプⅠ，Ⅱそれぞれに対して確率タスクTは以下のように異なる処理量a1，a2を持つものとする．

ただし，

a：確率タスクTの発生したときの処理量

p：確率タスクTの発生確率

とする．

（タイプⅠ）

確率タスクTが発生した場合にのみ発生する直後タスクであり，以下のようなタスクである．

○確率タスクTと制御依存関係がある直後タスク

○確率タスクTと制御依存関係のあるタスクと制御依存関係がある直後タスク．

このタイプに属する直後タスクが発生するのは，確率タスクTが発生する場合に限られるので，これらの直後タスクが発生したときには，必ず確率タスクTは発生している．このことより，直後タスクがタイプⅠに属するならば，この直後タスクに対する確率タスクTの処理量a1を次式のように定める．

$$a1 = a \qquad (4\text{-}1)$$

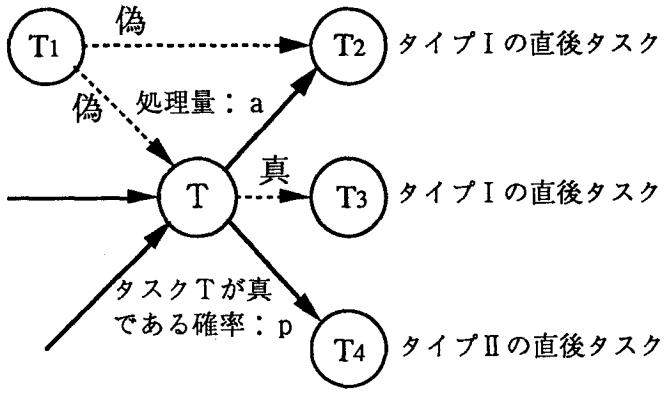
（タイプⅡ）

タイプⅠ以外の直後タスクである．つまり確率タスクTの発生とは独立に発生するタスクであり，確率タスクTとデータ依存関係のみがある．

このタイプに属する直後タスクは確率タスクTの発生とは関係がなく，確率タスクTが発生しない場合でも発生する．確率タスクTの発生しない場合の確率タスクTの処理量は0，確率タスクTが発生する場合には，処理量はaであることからタイプⅡの直後タスクに対する確率タスクT

の処理量ａ2を次式のように処理量の期待値とする．

$$a2 = a * p \qquad (4\text{-}2)$$



タイプＩの直後タスク
に対するＴの処理量：ａ

タイプⅡの直後タスク
に対するＴの処理量：ａ＊ｐ

図4-4　確率タスクの処理量

（2）　優先順位の決定法

本論文で提案するタスク多重割当法は，CP法やCP/MISF法と同様にリストスケジューリングの一種であり，１つのプロセッサに割当てることのできる複数のタスクが存在する場合，あらかじめタスクに付けられた優先順位に従い，最も優先順位の高いタスクをプロセッサに割当てる．

本割当法では，最長パス長が長いタスクに高い優先順位を与え，最長パス長が等しいタスクが複数存在する場合は，直後タスクの多いタスクほど高い優先順位を与える．しかし，後続タスクには確率タスクを含むため，終了タスクへ至る最長パス長を以下のように定める．

（最長パス長）

タスクTの直後タスクを，それらのタスクの発生する条件文の結果により次の3つのタイプに分類する．また，各タイプのタスクの最長パス長をL1，L2，L3とする．

① タスクTの結果にかかわらず発生する
② タスクTが真のとき発生する
③ タスクTが偽のとき発生する

タスクTの終了タスクへ至る最長パス長は

$$a + L_{MAX} * (P_{MAX}) + L_{MID} * (1 - P_{MAX}) \qquad (4\text{-}3)$$

となる．ただし

a：タスクTの処理量
LMAX：L1，L2，L3の最大値
PMAX：LMAXを与える直後タスクの発生確率
LMID：L1，L2，L3の中央値

本割当法における最長パス長算出の例を図4-5に示す．

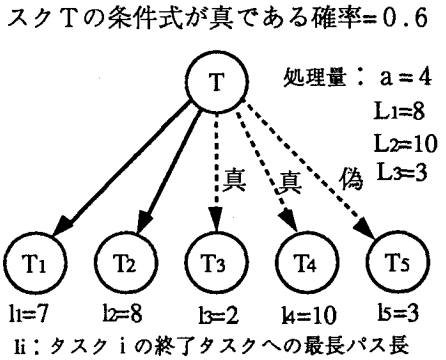


図4-5　直後タスクに確率タスクを含む場合の終了タスクへの最長パス長

（3）　割当てアルゴリズム

タスク多重割当法のアルゴリズムは，タスク間の排反関係の検出や優先順の決定を行う前処理部と，各タスクをどのプロセッサに割当てるかを決定する割当て部から構成される．各構成部の役割は以下のようなものである．

○前処理部
(1) 開始・終了ノードの生成
(2) 終了ノードへの最長パス長の算出
(3) 確率タスク間の排反関係の検出
(4) 確率タスクの処理量の算出

○割当て部
(1) 初期設定
(2) 処理可能タスクの選出
(3) プロセッサへのタスクの割当て
(4) 現時刻の更新
(5) プロセッサ状態の更新

以下，前処理部と割当て部について詳説する．

I　前処理部

前処理部には，ジョブ分割法から渡されるタスクに関する情報を基に，それらの情報を割当て部が割当てを行う際に利用できる形に変換する働きがある．具体的には以下に示す機能がある．

(1) 開始・終了タスクの生成

タスク集合に以下のように開始タスクと終了タスクを追加する．

・開始タスク

先行タスクを持たないタスクを検索し，これら全てのタスクを直後タスクとする処理量0のタスクを生成する．

・終了タスク

後続タスクを持たないタスクを検索し，これら全てのタスクを直前タスクとする処理量0のタスクを生成する．

(2) 終了タスクへの最長パス長の算出

各タスクに対し，終了タスクへ至る最長パス長を（4-3)を用いて算出する．

(3) 確率タスク間の排反関係の検出

4．2で述べたように，互いに発生条件の異なる確率タスクを検索し排反関係の検出を行う．

(4) 確率タスクの処理量

確率タスクTの各直後タスクに対して確率タスクTの処理量を

タイプⅠ：a1＝a　　（4-1）

タイプⅡ：a2＝a＊p　　（4-2）

とする．

Ⅱ　割当て部

割当て部では以下のアルゴリズムに従い，プロセッサへのタスクの割当てを行う．

以下の，アルゴリズムで用いられる記号を以下のように定義する．

○タスクに関する情報

Aj　：処理量

A1j　：タイプⅠの直後タスクに対する確率タスクjの処理量

A2j　：タイプⅡの直後タスクに対する確率タスクjの処理量

Pj　：確率タスクjの発生確率

○システム変数

time　：現在の時刻

○各プロセッサiに関する変数

processor_state_i：　プロセッサiの状態変数

“空き”，“稼働中”の2つの状態がある．

processor_time_i：　プロセッサiの完了時刻

○タスクjに関する変数

task_state_j　：タスクjの状態変数

“待機中”，“処理可”，“実行中”，“完了”の4つの状態がある．

task_time_j　：タスクjの完了時刻@m♯
task_state1_j　：タイプⅠの直後タスクに対する確率タスクjの状態変数“待機中”，“処理可”，“実行中”，“完了”の4つの状態がある．
task_time1_j　：タイプⅠの直後タスクに対する確率タスクjの完了時刻
task_state2_j　：タイプⅡの直後タスクに対する確率タスクjの状態変数“待機中”，“処理可”，“実行中”，“完了”の4つの状態がある．
task_time2_j　：タイプⅡの直後タスクに対する確率タスクjの完了時刻

(1) 初期設定

現時刻time := 0
・全タスクjに対し　task_state_j → “待機中”
・全プロセッサiに対し　processor_time_i := 0,
　　processor_state_i → “空き”
・開始タスクの task_state_0 → “完了” とする．

(2) 処理可能タスクの選出

・task_time_j ＝ timeであれば，task_state_j → “完了”
　ただし，確率タスクであれば，
　task_time1_j ≦ time ならば，task_state1_j → “完了”
　task_time2_j ≦ time ならば，task_state2_j → “完了”
　とする．
・全ての直前タスクのtask_state ＝ “完了”となったタスクiに対し，
　task_state_i → “処理可”とする．
・終了タスクのtask_state ＝ “処理可”となれば，プログラムを終了する．

(3) プロセッサへのタスクの割当て

全てのプロセッサに対して“空き”か“稼働中”を調べ，
(a) processor_state_i ＝ “空き”の場合，
　・task_state ＝ “処理可”であるタスクで最も優先順位の高いタスクを割当てる．

・プロセッサiに割当てられたタスクjが確率タスクでなければ，

task_time_j := time＋ Aj

processor_time_i := time＋ Aj

processor_state_i → “稼働中”

とする．

・プロセッサiに割当てられたタスクjが確率タスクであれば，

task_time1_j := time＋ A1j

task_time2_j := time＋ A2j

task_state1_j → “実行中”

task_state2_j → “実行中”

processor_time_i := processor_time_i＋ A2j

processor_state_i → “稼働中”

とする．

・割当てられたタスクと互いに排反で task_state_k= “処理可” の確率タスクkがあれば多重割当てを行う．

多重割当てされたタスクkおよびプロセッサiの変数を

task_time1_k := time＋ A1k

task_time2_k := time＋ A2k

task_state1_k → “実行中”

task_state2_k → “実行中”

processor_time_i := processor_time_i＋ A2k

とする．

(b) processor_state_i ＝ “稼働中” の場合，

task_state ＝ “処理可” の確率タスクが，プロセッサiで実行中のタスクと互いに排反ならば割当てる．（多重割当て）

割当てられたタスクkおよびプロセッサiの変数を

task_time1_k := time＋ A1k

task_time2_k := time＋ A2k

processor_time_i := processor_time_i＋ A2k

とする．

(4) 現時刻の更新

timeを processor_time, task_time, task-time_1, task_time_2の最小値に更新する.

(5) プロセッサ状態の更新

processor_time_i ＝timeであるプロセッサiに対し

processor_state_i →“空き”

とする.

(2) へ

4.4　タスク多重割当法の評価

本節では，タスク多重割当法と従来の割当法を比較するために，タスク多重割当法と，タスクの排反性を考慮せずに全てのタスクが常に発生すると仮定しCP/MISF法を用いて割当てた場合とを比較する.

図4-6に示す依存関係を持つタスク集合を対象に，プロセッサ台数

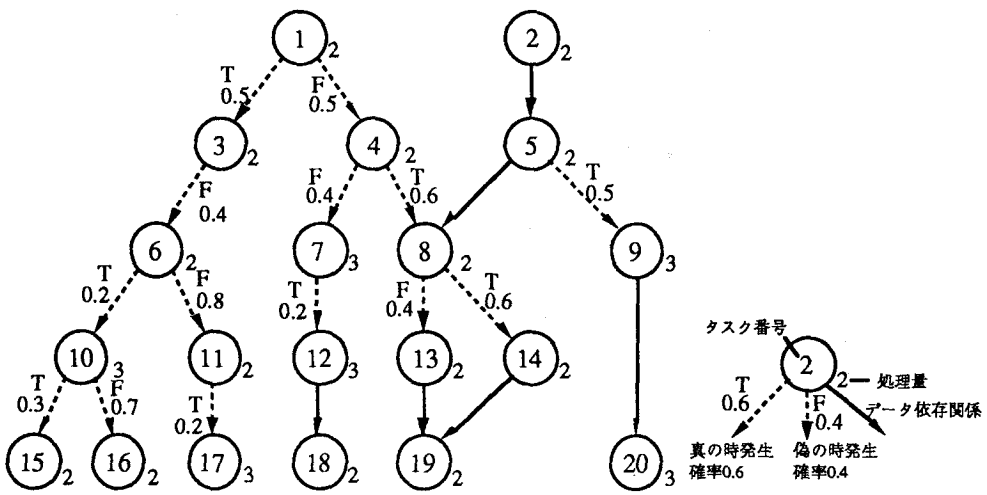


図4-6　割当て対象例

が3台という条件の下で，タスク多重割当法とCP/MISF法を適用した割当て結果を表4-1に示す．表中の斜体文字は多重割当てされたタスクを表している．表4-1からわかるように，CP/MISF法では全てのタスクが各プロセッサに均一に割当てられている．それに対し，タスク多重割当法では，プロセッサ1に対して多くのタスクが割当てられている．しかしながら，これは互いに排反なタスクが多重割当てされたためであり，実際の処理系列ではこれらのタスクが全て発生するわけではない．そこで，実際に処理系列個々に対して，タスク多重割当法による割当て結果とCP/MISF法による割当て結果の処理終了時間を比較すると表4-2のようになる．

表4-2からわかるように，実際の処理系列ではタスク多重割当法の方がCP/MISF法よりも処理時間が短くなる．

表4-1　割当て結果の比較

プロセッサ	割当てられたタスク	
	多重割当法	CP/MISF法
1	2,5,*6*,*7*,8,*10*,*11*,*12*,*13*,*14*,*15*,*16*,*17*	1,4,7,11,12,15,19
2	1,*3*,4,9,19	2,3,6,9,13,18,20
3	20,18	5,8,10,14,16,17

(斜体は多重割当てされたタスクを表す)

表4-3　割当て結果の比較（プロセッサ数10）

プロセッサ	割当てられたタスク	
	多重割当法	CP/MISF法
1	2,5,*6*,*7*,8,*10*,*11*,*12*,*13*,*14*,*15*,*16*,*17*	1,4,7,12,18
2	1,*3*,4,9,19	2,3,6,10,15
3	20	5,8,11,17
4	18	9,20
5		13,19
6		14,16

(斜体は多重割当てされたタスクを表す)

表4-2　処理終了時間の比較

実行されるタスク系列	終了時間	
	多重割当法	CP/MISF法
1,2,3,5,18,19,20	6	9
1,2,3,5,6,10,15,18,19,20	11	11
1,2,3,5,6,10,16,18,19,20	11	11
1,2,3,5,6,11,17,18,19,20	11	11
1,2,3,5,6,11,18,19,20	8	11
1,2,3,5,6,9,10,15,18,19,20	11	14
1,2,3,5,6,9,10,16,18,19,20	11	14
1,2,3,5,6,9,11,17,18,19,20	11	14
1,2,3,5,6,9,11,18,19,20	10	14
1,2,3,5,9,18,19,20	10	12
1,2,4,5,7,12,18,19,20	12	12
1,2,4,5,7,18,19,20	7	9
1,2,4,5,7,9,12,18,19,20	12	12
1,2,4,5,7,9,18,19,20	10	10
1,2,4,5,8,13,18,19,20	10	11
1,2,4,5,8,14,18,19,20	10	10
1,2,4,5,8,9,13,18,19,20	10	12
1,2,4,5,8,9,14,18,19,20	10	10

表4-4　処理終了時間の比較（プロセッサ数１０）

実行されるタスク系列	終了時間	
	多重割当法	CP/MISF法
1,2,3,5,18,19,20	6	4
1,2,3,5,6,10,15,18,19,20	11	11
1,2,3,5,6,10,16,18,19,20	11	11
1,2,3,5,6,11,17,18,19,20	11	11
1,2,3,5,6,11,18,19,20	8	8
1,2,3,5,6,9,10,15,18,19,20	11	11
1,2,3,5,6,9,10,16,18,19,20	11	11
1,2,3,5,6,9,11,17,18,19,20	11	11
1,2,3,5,6,9,11,18,19,20	10	10
1,2,3,5,9,18,19,20	10	10
1,2,4,5,7,12,18,19,20	12	12
1,2,4,5,7,18,19,20	7	9
1,2,4,5,7,9,12,18,19,20	12	12
1,2,4,5,7,9,18,19,20	10	10
1,2,4,5,8,13,18,19,20	10	10
1,2,4,5,8,14,18,19,20	10	10
1,2,4,5,8,9,13,18,19,20	10	10
1,2,4,5,8,9,14,18,19,20	10	10

つぎに，同じタスク集合に対して，タスク多重割当法とCP/MISF法の各方式で最適な割当てを行う際に必要となるプロセッサ台数の比較検討を行う．ここでは，プロセッサ台数に制限がないという条件で割当てを行い，必要としたプロセッサ台数と実際の処理系列での処理終了時間を比較する．

図4-6のタスク集合に対して割当てを行った結果，タスク多重割当法とCP/MISF法のそれぞれに対し表4-3に示す割当て結果を得ることができた．また，実際の処理系列における処理終了時間は表4-4の様になる．これらの表からわかるように，タスク多重割当法とCP/MISF法では実際に処理をする際の終了時間はほぼ同じであるにもかかわらず，CP/MISF法で必要となるプロセッサ台数6台に対して，タスク多重割当法では4台のプロセッサとなっている．

以上の2例からわかるように，タスク多重割当法は，排反性を考慮しないCP/MISF法に比べ，プロセッサ台数が同じ場合にはより短い処理時間となる割当てを行うことができ，また，同程度の処理時間とするためのプロセッサ台数がより少なくできることがわかる．これは，4．3で述べたように互いに排反なタスクを同一のプロセッサに割当てたことにより，実際の処理実行時にプロセッサのアイドル状態の発生を抑制することができるためである．

4.5　結言

本章では，確率タスク間の排反関係に注目し多重割当てを行う“タスク多重割当法”の提案を行った．タスク多重割当法はリストスケジュールの一種である．

その発生が確率的に決定される確率タスク間には，いかなる実行系列においても，共に発生することのないタスクが存在する．これを確率タスク間の排反関係という．これら互いに排反なタスクは同一のプロセッサに割当てる“多重割当て”を行っても実行時には共に発生することはないためプロセッサを巡り競合することはない．

そこで，タスク多重割当法ではタスク間の依存関係である制御依存関係とデータ依存関係の内，制御依存関係に注目しタスク間の排反関係の

検出を行い，互いに排反なタスクを同一のプロセッサに割当てる，排反タスクの多重割当てを行う．

タスク多重割当法では，確率タスクの発生条件とその確率タスクの直後タスクの発生条件が異なる場合には，確率タスクの処理量に発生確率を掛け合わせたものを処理量とする．

また，タスク多重割当法では，後続タスクを常に発生するタスクと条件が真の時に発生するタスク，ならびに，偽のときに発生するタスクの3種類のタスクに分類し，各タイプのタスクの発生確率と各タイプの最長パス長を用い，最長パス長の期待値を算出することで，優先順位の決定を行う．

さらに，本章ではこのタスク多重割当法と，タスクの排反性を考慮せずに全てのタスクが常に発生すると仮定してCP/MISF法を適用した場合の割当て結果を比較し，タスク多重割当法の方が，同程度の処理終了時間を得るために必要となるプロセッサ台数が少ないことを，また，プロセッサ台数に制限がある場合にはより短い処理終了時間となることを示した．

第5章

先行制御方式

5.1　緒言

並列処理技術は，大規模なジョブを高速に処理する方法として注目され，画像処理[KAWA 88][MITS 85][TADA 88]や行列計算等の分野において広く用いられている．これらの中で，モンテカルロ・シミュレーションのように並列性を有しているが，乱数によって次の処理を決定しながら進むジョブの並列処理に関しては次のような研究がなされている．文献[INAM 85][SATA 83]では時刻駆動型，文献[NAKA 82]では事象駆動型の待ち行列網シミュレータが提案されている．文献[YOSH 87]では制御用のメッセージを用いることにより同期をとる方式，また，文献[JEFF 85a][JEFF 85b]及び[SATO 86]では一次的な矛盾を許容しながらも最終的には矛盾を解消する方式の提案が行われている．これらのプロセッサ間同期法のうち，一時的な矛盾を許容する先行制御方式は，他の方式に比べプロセッサの利用率を高くすることができるという利点がある．

先行制御方式を含めプロセッサ間同期方式を用いた並列処理システムにおけるプロセッサ間通信には，確率的要素が含まれることが知られている．しかしながら，プロセッサ間通信に存在するこの確率的要素を数学的に取り扱えるモデル化法は従来存在しなかった[CHAN 78][CHAN 79][CHAN 81].

本章の目的は先行制御方式におけるプロセッサ間同期を数学的に取り扱い，処理速度低下の主な原因となる矛盾状態発生率を求めることであ

る．そこで，まず並列処理システムのプロセッサ間同期を取り扱う概念として従来提案されてきたメッセージ・パッシング[HEWI 77][YONE 85]を拡張し，実行する処理を確率的に決定しながら進行するスケジュール不可能なジョブを並列処理する際のプロセッサ間同期にも適用できるようにする．さらに，これを用いて先行制御方式の解析を行い[WATA 86][KOBA1 87][KOBA1 88a][KOBA1 88b][KOBA1 88c][KOBA1 91a]，オーバヘッドに関係する矛盾状態発生率を小さくするための条件を示す．

5.2　先行制御方式の同期アルゴリズム

先行制御方式では，以下に示すアルゴリズムで処理を行う．

① 各プロセッサは他のプロセッサとまったく独立に処理を行う．
② 送信側プロセッサはメッセージ送信時に自分自身の論理時刻を付加し送信する．
③ 受信側プロセッサはメッセージを受け取ると，メッセージの持つ論理時刻ｔｍと現在の論理時刻ｔを比較する．

◎ｔｍ≧ｔのとき

論理時刻がｔｍに達するまでこのメッセージの処理を行わない．

◎ｔｍ＜ｔのとき

受信側プロセッサは処理を進めすぎたこととなり矛盾が発生する．この場合，受信側プロセッサは進みすぎた処理を取り消し，論理時刻をｔｍまで戻し（キャンセル処理），処理を再開する．

図５-１は先行制御を行った際のプロセッサの実時刻と論理時刻の関係を示したものである．図５-１において，送信側プロセッサＳは実時刻Ｔにメッセージを送り，このメッセージの持つ論理時刻はｔｓである．このときの受信側プロセッサＲの論理時刻はｔｒ＞ｔｓであり，矛盾となる．そこでプロセッサＲはその状態を論理時刻ｔｓのときの状態に戻し，矛盾を解消し，再処理を行う．

先行制御方式では，受信側プロセッサはメッセージを受け取るまで送られてくる時刻を知ることはなく，メッセージが送られて初めて矛盾が

発生したかどうかの判断を行う．このように先行制御方式では，処理過程に一時的な矛盾は発生するが，矛盾が発生するたびにキャンセル処理によりその矛盾を解消することで，最終的な処理過程から矛盾を排除している．

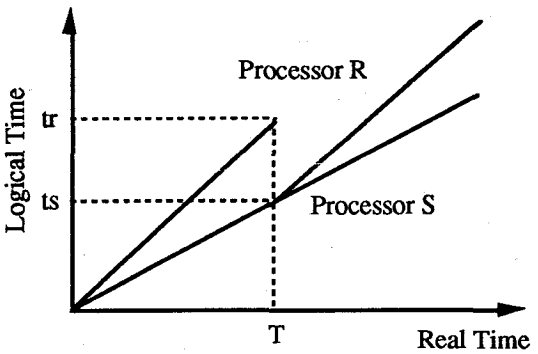


図5-1　先行制御方式

5.3　拡張メッセージ・パッシングの提案

5.3.1　メッセージ・パッシングの分類

並列処理システムでジョブを実行する場合，各プロセッサはイベント間の半順序関係を満たさねばならないために，メッセージの送受信で互いに関係を持ちながら処理を進めていく．そのため，受信側プロセッサではメッセージが送られてくるのかどうか，また送られてきたときにはそのときの受信側プロセッサの論理時刻が問題となる．この2点に注目すると，並列処理システムにおけるプロセッサ間の通信はメッセージ・パッシングの概念を拡張することで取り扱うことができ，スケジュール可能なジョブと不可能なジョブにおけるプロセッサ間通信を明確に区別することが可能となる．

メッセージ・パッシングは，従来オブジェクト指向型言語などの分野でオブジェクト間の通信を取り扱うために使われてきたものであり，メッセージを送る送信オブジェクトと，受け取る受信オブジェクトの関係

を取り扱うものである．このメッセージ・パッシングは従来ｐａｓｔ型，ｎｏｗ型，ｆｕｔｕｒｅ型の3種類に分類されてきた．ｐａｓｔ型はメッセージを送受信することなく相互に関係を持つことのないプロセッサ間の関係を，ｎｏｗ型とｆｕｔｕｒｅ型は予め決定されている論理時刻にメッセージの送受信が行われる場合のプロセッサ関係をモデル化している．しかし，既存のタイプのメッセージ・パッシングはメッセージが明らかに送られてくる場合と明らかに送られてこない場合のみを想定しており，メッセージの送受信が確率的に起こる場合は全く考慮されていない．また，メッセージを受け取るまでに受信側オブジェクトが行わなければならないタスクについては，その有無は取り扱えるが，その量は取り扱えない．したがって，既存のタイプのメッセージ・パッシングだけでは並列処理システムのプロセッサ間通信を完全に取り扱うことはできない．そこで，メッセージの送受信の有無や，送受信時の論理時刻に確率的要素を取り入れることによって従来のメッセージ・パッシングを拡張したｕｎｋｎｏｗｎ型メッセージ・パッシングの提案を行う．

並列処理システムの各プロセッサ間関係は図5-2のような横軸に実時刻，縦軸に論理時刻を取ったグラフで表すことができ，メッセージ・パッシングは以下に示す2つの観点から分類することができる．

① プロセッサＡがプロセッサＢに対してメッセージを送るかどうか．
② メッセージを送る場合に，プロセッサＡの送ってくるメッセージの論理時刻．

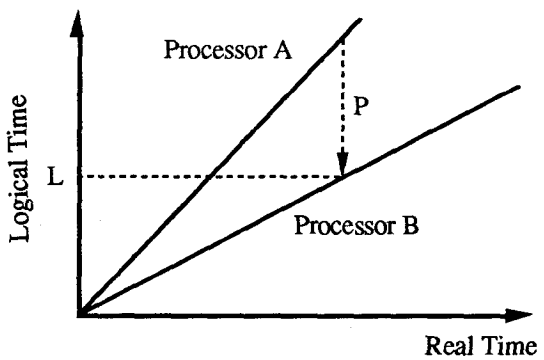


図5-2 メッセージパッシング

表5-1はメッセージ・パッシングを，メッセージが送られる確率Pとその論理時刻Lで分類したものである．表5-1において“ｋｎｏｗｎ”は，プロセッサBがプロセッサAからメッセージが送られる以前にメッセージが送られる論理時刻がわかっていることを意味する．

表5-1　メッセージパッシングの分類

P	L	TYPE
0	———	past
1	known	now, future
	unknown	unknown-I
(0,1)	known	unknown-II
	unknown	unknown-III

5.3.2　ｋｎｏｗｎ型メッセージ・パッシング

ｋｎｏｗｎ型メッセージ・パッシングは不確定要素を含まないメッセージ・パッシングで既存のタイプのメッセージ・パッシングである．ｋｎｏｗｎ型メッセージ・パッシングは，具体的にはｐａｓｔ型，ｎｏｗ型，ｆｕｔｕｒｅ型の3種類に分類できる．以下にこれら3種類のメッセージ・パッシングについて述べる．

(1)　ｐａｓｔ型メッセージ・パッシング

ｐａｓｔ型メッセージ・パッシングでは送信確率Ｐ＝0である．プロセッサAとプロセッサBが同期をとった後，プロセッサAからプロセッサBに対しメッセージが送られることはないので，プロセッサBはプロセッサAからのメッセージを待つことなく，処理を進めることができる．

(2)　ｎｏｗ型およびｆｕｔｕｒｅ型メッセージ・パッシング

このタイプのメッセージ・パッシングでは，受信側プロセッサBは送信側プロセッサAがメッセージを必ず送り（Ｐ＝1），かつ（論理時刻で）いつ送るかもわかっている（Ｌ：ｋｎｏｗｎ）．ｎｏｗ型とｆｕｔｕｒｅ型は以下のような違いがある．

ｎｏｗ型はLが現在の受信側プロセッサの論理時刻と一致している，つまりプロセッサBはメッセージが送られてくるまで論理時刻を更新しない．サブルーチンコールやファンクションコールなどがこれに当る．

一方，ｆｕｔｕｒｅ型メッセージ・パッシングではLは受信側プロセッサの現在の論理時刻よりも大きい．つまりプロセッサBは，プロセッサAからメッセージが送られてくるまでの間に何らかの処理を行い論理

時刻の更新を行う．ｆｕｔｕｒｅ型メッセージ・パッシングには，行列計算を行うアレイプロセッサにおけるプロセッサ間通信などがある．

5.3.3　ｕｎｋｎｏｗｎ型メッセージ・パッシング

ｕｎｋｎｏｗｎ型メッセージ・パッシングは細かく分けると３種類に分類できる．

(1)　ｕｎｋｎｏｗｎ－Ⅰ型メッセージ・パッシング

メッセージの送受信確率 Ｐ＝１ であるが，送受信時の受信側プロセッサの論理時刻は確定せず，ある確率分布に従う（Ｌ：ｕｎｋｎｏｗｎ）．つまり，プロセッサＡからメッセージが送られてくることは確定しているが，プロセッサＢはメッセージがいつ送られてくるのかわからない．

(2)　ｕｎｋｎｏｗｎ－Ⅱ型メッセージ・パッシング

プロセッサＡからメッセージが送られてくるかどうかはわからないが（０＜Ｐ＜１），プロセッサＢはメッセージがもし送らてくるならば，その時刻を知っている（Ｌ：ｋｎｏｗｎ）．

(3)　ｕｎｋｎｏｗｎ－Ⅲ型メッセージ・パッシング

プロセッサＡからメッセージが送られてくるかどうかわからないし（０＜Ｐ＜１），もし送られてくるとしても，その時刻もわからない（Ｌ：ｕｎｋｎｏｗｎ）．

スケジュール可能なジョブを並列処理システムで実行した場合のプロセッサ間通信は，いつ起こるかがメッセージの受信以前にわかっているため，ｎｏｗ型，ｐａｓｔ型，ｆｕｔｕｒｅ型のメッセージ・パッシングとなる．

一方，スケジュール不可能なジョブを並列処理システムで実行した場合は，メッセージが送られてくるのかどうか，またいつ送られてくるのかわからないため，ｋｎｏｗｎ型メッセージ・パッシングだけではなくｕｎｋｎｏｗｎ型メッセージ・パッシングを含む．

ｕｎｋｎｏｗｎ型メッセージ・パッシングを含むジョブを並列処理システムで実行した場合，送信プロセッサＡと受信プロセッサＢの論理時刻により矛盾状態が発生する場合がある．ここではプロセッサＡがプロセッサＢにメッセージを送るときの実時刻をＴ，プロセッサＡの論理時

刻をｔA，プロセッサＢの論理時刻をｔBとする．以下大文字Ｔは実時刻を，小文字ｔは論理時刻を表すものとする．またプロセッサ間通信は瞬時に行われるものとし，遅延はないものとする．

(1)　ｔA≧ｔBの場合

プロセッサＡからメッセージが送られてきたときには，受信側プロセッサＢは送信側プロセッサＡよりも論理時刻が遅れている．この場合プロセッサＢは，論理時刻がｔAとなる実時刻ＴBまで待ち，プロセッサＡから送られてきたメッセージに対する処理を行う．（図5-3）

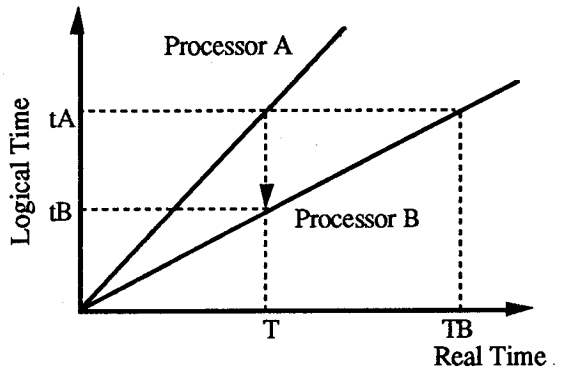


図5-3 無矛盾状態

(2)　ｔA＜ｔBの場合

プロセッサＢにはプロセッサＡから論理時刻がｔAであるメッセージが送られてくる．この場合，プロセッサＢでは論理時刻がｔAのときにこのメッセージに対する処理を行わなければならない．しかしながら，プロセッサＢの論理時刻がｔAとなるときの実時刻ＴBはＴB＜Ｔであり，メッセージが送られる以前を指す．つまり，過去ＴBにおいてこのメッセージに対する処理が行われなければならなかったことがわかった状態となる．この状態を論理矛盾（簡単に矛盾）という．（図5-4）

矛盾はイベントの発生順序が正しくないときに生じる状態である．そのためジョブの処理を終了した時点でその過程に矛盾があれば，処理シ

ステムの出した結果は正しくなくなる．

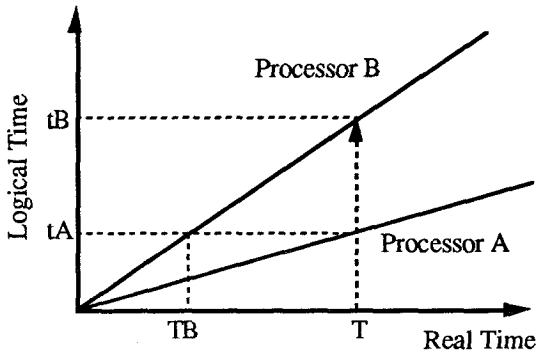


図5-4 矛盾状態

5.4　先行制御方式の処理効率解析

本章では，プロセッサ間同期法として先行制御方式を取り上げ，5．3で提案した拡張メッセージ・パッシングを用いて解析を行い，評価を行う．

並列処理システムの評価基準として処理速度改善比があげられる．処理速度改善比は並列処理システムが逐次処理システムと比べどれだけ高速に処理ができるかを示す尺度である．

逐次処理システムと並列処理システムで論理時刻tに達する実時刻を比べると，図5-5からわかるように逐次処理システムでは明らかにこの実時刻はTであり，処理速度はVs=t／Tである．一方，m台のプロセッサからなる並列処理システムでは，あるプロセッサがいかに速く処理を行ったとしても処理システム全体の論理時刻は最も遅れているプロセッサkの論理時刻となるため，並列処理システムとして論理時刻tに達するのは実時刻Tkのときである．したがってこの場合，並列処理システムとしての処理速度はVp=t／Tkとなる．そこで，並列処理シ

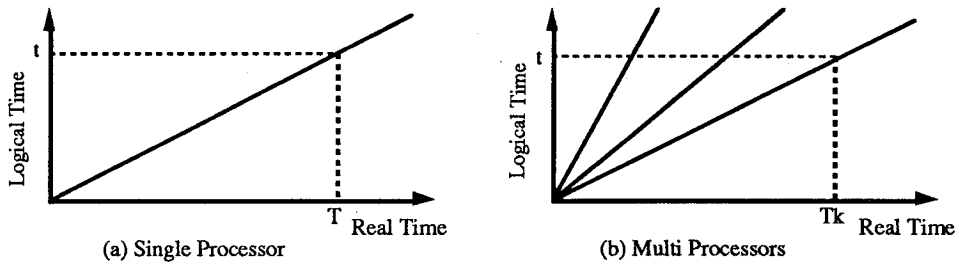


図5-5 逐次処理対並列処理

ステムの処理速度改善比Riを，

$$Ri = \frac{Vp}{Vs} = \frac{\min(V1, \cdots, Vm)}{Vs} = \frac{T}{\max(T1, \cdots, Tm)} \qquad (5-1)$$

と定義する．つまり処理速度改善比Riは，同一のジョブを逐次処理システムで実行した場合に比べ，並列処理システムでは1／Riの実時間で終了することを意味する．

また，並列処理するための分散化オーバヘッドがなければ，

$$T = T1 + \cdots\cdots + Tm \qquad (5-2)$$

となるため，Riは　$T1 = T2 = \cdots\cdots = Tm = T/m$ のときに

最大値　$Ri\_max = m$．

また，　$Tj = T$，$Ti = 0$　$(i \neq j)$　のときに

最小値　$Ri\_min = 1$．

となる．このように分散化オーバヘッドがないと仮定した場合の処理速度改善比を理想処理速度改善比と呼びRi_idealで表す．

しかしながら，現実の並列処理システムでは分散化オーバヘッドがあり，

$$T + \alpha = T1 + \cdots\cdots + Tm \quad (\text{但し，} \alpha > 0)$$
$$T < T1 + \cdots\cdots + Tm$$

であるため，実処理速度改善比Ri_realは，

$$Ri\_real < Ri\_ideal.$$

となる．

理想処理速度改善比Ri_idealと実処理速度改善比Rirealの比d

$$d = 1 - Ri\_real / Ri\_ideal.$$

が分散化オーバヘッドによる処理速度低下であると考えられる．

実際の並列処理システムを構築する際には，この分散化オーバヘッドによる処理速度低下dを少なくすることが1つの目標となる．また，dを直接求めることができない場合には，dと関係の深い要素について検討を行い並列処理システムの設計を行う．

先行制御方式では，矛盾状態の生起はそれまで余分な処理を行っていたことを意味し，矛盾解消のためにキャンセル処理を行わなければならない．したがって，その分処理速度は低下する．以下では，dに最も深く関係する矛盾状態発生率を解析により求め，先行制御方式の処理速度改善について検討する．

・　矛盾状態発生率

$$(\text{矛盾状態発生率}) = \frac{(\text{矛盾状態発生回数})}{(\text{メッセージ送信回数})}$$

解析対象となるモデルを以下のように仮定し，このモデルの矛盾状態発生率を求める．

① システムは送信側プロセッサSと受信側プロセッサRの2台のプロセッサからなる．
② 送信側プロセッサが1イベント終了毎にメッセージを送る確率はPs=0．5である．
③ 各プロセッサの1イベントに要する実時間は，送信側プロセッサSにおいては平均Rs=1／λs，受信側プロセッサRにおいては平均Rr=1／λrの指数分布に従う．

④　各プロセッサが1イベントで更新する論理時間は，送信側プロセッサSにおいては平均Ls＝1／μs，受信側プロセッサRにおいては平均Lr＝1／μrの指数分布に従う．
⑤　メッセージの伝搬遅延はない．

以上のようなモデルに対し，以下のような手順で矛盾状態発生率を求めた．

送信側プロセッサSは平均1／Ps個のイベント毎にメッセージを送る．つまり，実時間でTs＝（1／Ps）×Rs毎にメッセージを送る．したがって，この場合プロセッサ間の通信は必ずメッセージが送られるが，メッセージの持つ論理時刻が予め決定されていないｕｎｋｎｏｗｎ－Ｉ型となっている．そこで，ｕｎｋｎｏｗｎ－Ｉ型で問題となるメッセージの持つ論理時刻の確率密度関数を求める．実時間Tsの間に送信側プロセッサSにK個のイベントが発生する確率PsK（Ts）は，平均到着時間Ls＝1／λsの指数到着間隔であるポアソン到着過程となるから，

$$P_{sK}(T_s) = \frac{(\lambda_s T_s)^K}{K!} \exp(-\lambda_s T_s) \qquad (5\text{-}3)$$

となる．また，同様にプロセッサRでL個のイベントが発生する確率PrL（Ts）は，

$$P_{rL}(T_s) = \frac{(\lambda_r T_s)^L}{L!} \exp(-\lambda_r T_s) \qquad (5\text{-}4)$$

となる．

したがって，実時間Ts秒の間にプロセッサSでK個，プロセッサRでL個のイベントが起こる確率P（K,L¦Ts）は，

$$P(K,L\,|\,T_s) = P_{sK}(T_s) \times P_{rL}(T_s) \qquad (5\text{-}5)$$

となる．

また，プロセッサSにおいてイベントがK個であるときの論理時間の分布の密度関数psK（t）は，Kステージアーラン分布となり，

$$psK(t)=\frac{K\ \mu s(K\ \mu s\ t)^{K-1}}{(K-1)!}exp(-K\ \mu s\ t) \qquad (5\text{-}6)$$

となる．このpsK（t）がunknown－I型メッセージ・パッシングにおけるメッセージの論理時刻tの確率密度関数である．また，同様にプロセッサRでイベント数がL個であるときの論理時間の分布の密度関数prL（t）は，

$$prL(t)=\frac{L\ \mu r(L\ \mu r\ t)^{L-1}}{(L-1)!}exp(-L\ \mu r\ t) \qquad (5\text{-}7)$$

となる．

図5-1からわかるようにtr＞tsのときに矛盾となる．プロセッサS，プロセッサRのイベント数がそれぞれK，Lであるときに矛盾状態となる確率PKL（tr＞ts）は

$$PKL(tr>ts)=\int_0^{\infty}\{psL(\tau)\int_{\tau}^{\infty}prK(x)\,dx\}d\tau \qquad (5\text{-}8)$$

となる．したがって，矛盾状態発生率Pcは，

$$Pc=\sum_{L=1}^{\infty}\sum_{K=1}^{\infty}\{P(K,L\,|\,Ts)\times PKL(tr>ts)\} \qquad (5\text{-}9)$$

となる．

5.5　数値例及び検討

まず，矛盾状態発生率Ｐcを一意対応に決定できるパラメータを求める．矛盾状態発生率Ｐcは明らかに１イベント当りの平均実時間Ｒs，Ｒr及び平均論理時間Ｌs，Ｌrによって決定できる，しかし，図５-６からわかるように，矛盾状態発生率ＰcはＲrやＲsの絶対値だけではなく，その比（Ｒr／Ｒs）によって決定されることがわかる．また，図５-７からわかるように，（Ｌs／Ｌr）によっても決定されることがわかる．したがって，矛盾状態発生率Ｐcは（Ｒr／Ｒs），（Ｌs／Ｌr）の２つのパラメータを決定すると一意に決定できる．図５-８はｘ軸に（Ｌs／Ｌr），ｙ軸に（Ｒr／Ｒs），ｚ軸に矛盾状態発生率Ｐcをとり３次元空間上に図示したものである．

次に，この解析対象である並列処理システムにおいて，理想処理速度改善比が最大値２であるという条件の下で，これら２つのパラメータ（Ｒr／Ｒs），（Ｌs／Ｌr）と矛盾状態発生率Ｐcとの関係について検討する．

５．４で述べたように理想処理速度改善比Ｒiは分散化オーバヘッドがないと仮定したときの送信側プロセッサの処理速度である理想処理速度Ｖs＝Ｌs／Ｒsと受信側プロセッサの理想処理速度Ｖr＝Ｌr／Ｒrが等しいときに最大値２となる．したがって，

$$Vs = Vr$$

$$\frac{Ls}{Lr} \times \frac{Rr}{Rs} = 1 \qquad (5\text{-}10)$$

すなわち，式（５-１０）を満たすとき処理速度改善比は最大値Ｒi＝２となる．

理想処理速度改善比Ｒi＝２のときに，２つのパラメータ（Ｒr／Ｒs），（Ｌs／Ｌr）と矛盾状態発生率Ｐcの関係を調べる．

Ｒi＝２の条件の下で，（Ｒr／Ｒs）と矛盾状態発生率Ｐcとの関係を示したものが図５-９であり，（Ｌs／Ｌr）と矛盾状態発生率Ｐcとの関係を示したものが図５-１０である．これらのグラフより，Ｒi＝２の条件

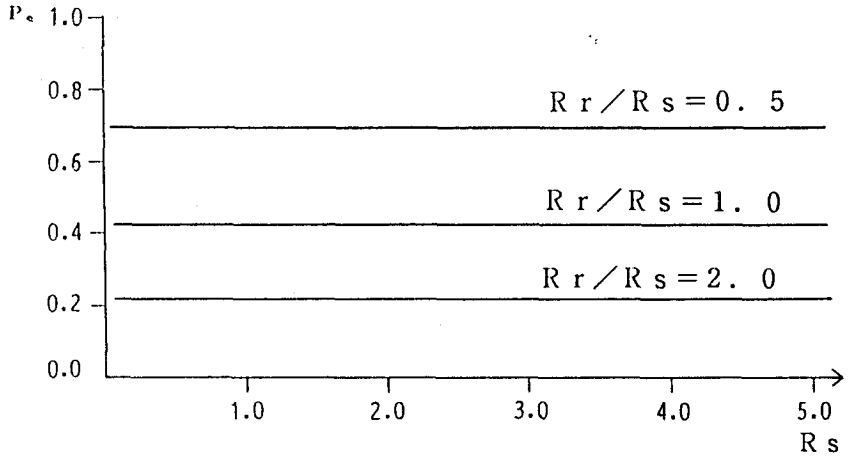


図5-6　送信側平均実時間Ｒｓ　ｖｓ．矛盾状態発生率Ｐｃ
（Ｌｓ＝Ｌｒ＝1．0）

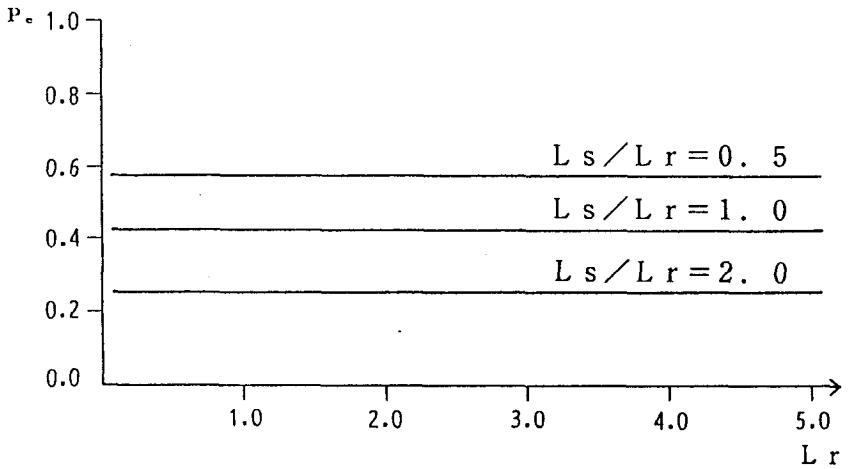


図5-7　受信側平均論理時間Ｌｒ　ｖｓ．矛盾状態発生率Ｐｃ
（Ｒｓ＝Ｒｒ＝1．0）

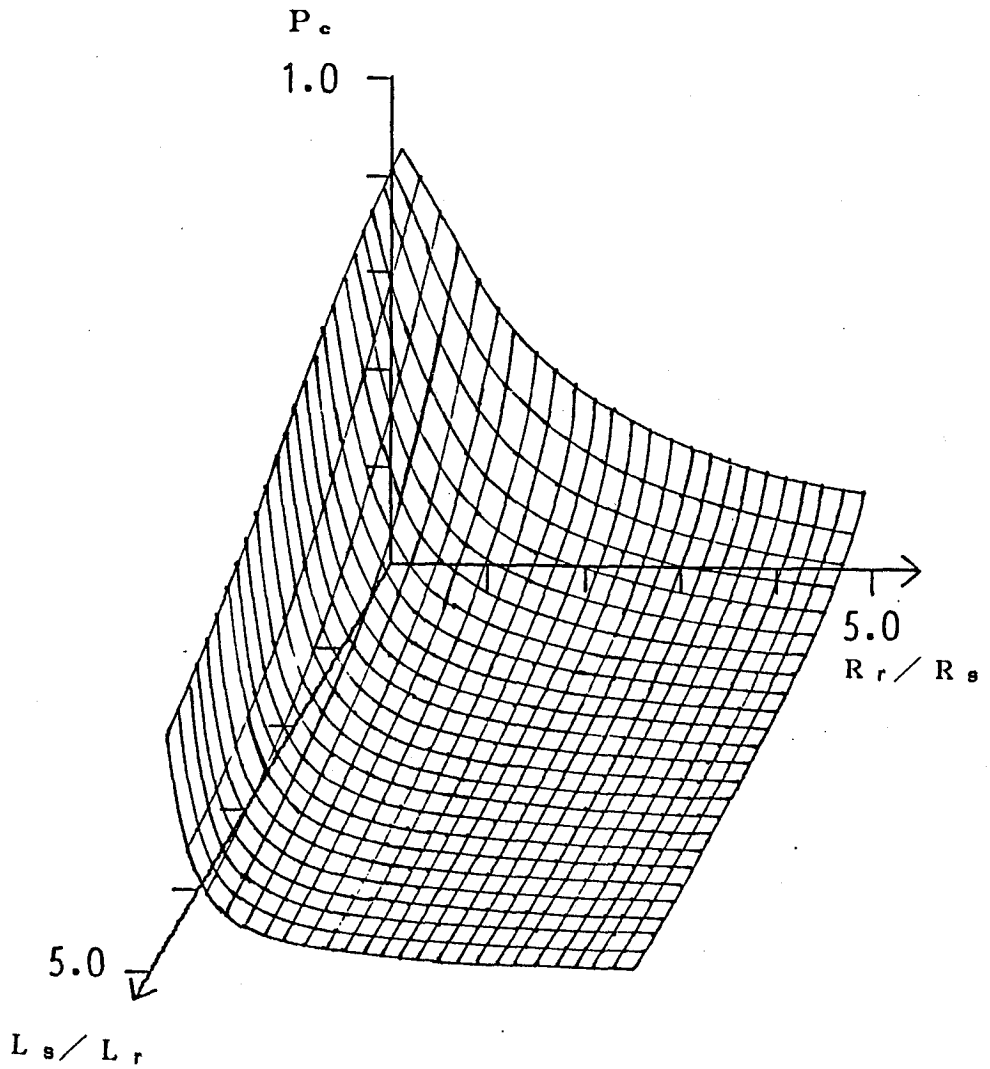


図5-8　論理時間比Ls／Lr
vs. 実時間比Rr／Rs
vs. 矛盾状態発生率Pc

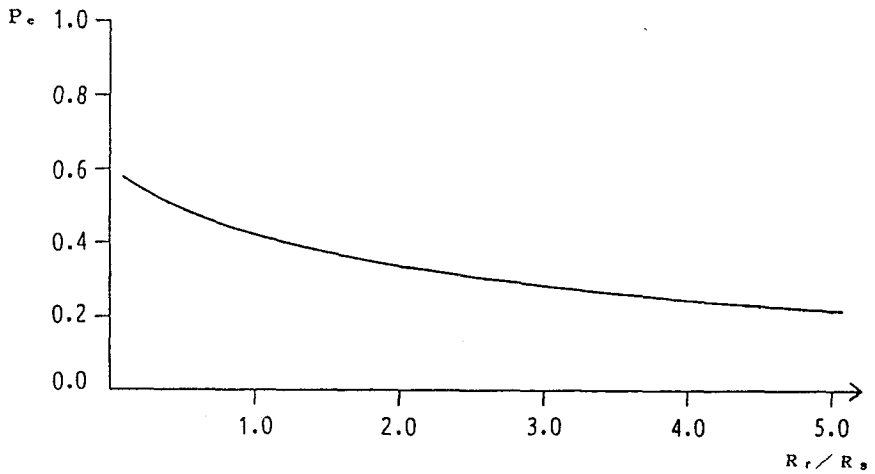


図５-９　実時間比Ｒｒ／Ｒｓ　ｖｓ．矛盾状態発生率Ｐｃ
（理想処理速度改善比　２．０）

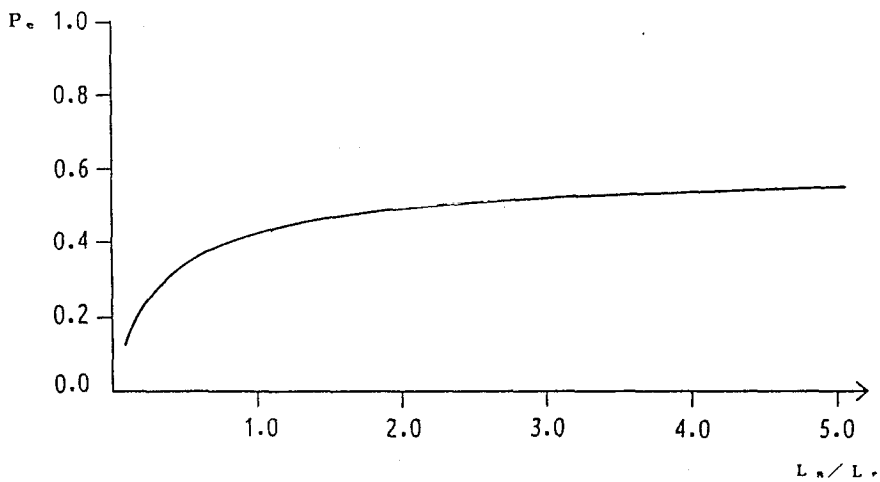


図５-１０　論理時間比Ｌｓ／Ｌｒ　ｖｓ．矛盾状態発生率Ｐｃ
（理想処理速度改善比　２．０）

の下では，（Rr／Rs）を大きく，（Ls／Lr）を小さくすることによって矛盾状態発生率を低減できることがわかる．

上記のようになる理由を定性的に検討する．プロセッサ間のメッセージの送信は，送信側プロセッサにおいては1イベント終了ごとに確率Psでなされる．イベントに要する時間より十分長い時間において，送信側プロセッサと受信側プロセッサの処理速度が等しい場合でも，図5-11に示すように実時刻更新量比（Rr／Rs）が大きく，論理時刻更新量比（Ls／Lr）が小さいときには，メッセージの送信時には受信側プロセッサが遅れていることが多い．そのため矛盾の発生が少なくなると考えられる．一方それに対し，実時刻更新量比（Rr／Rs）が小さく，論理時刻更新量比（Ls／Lr）が大きいときには，図5-12に示すように処理の開始後一番初めに送られるメッセージは矛盾の発生を引き起こすが，その後のメッセージの送信時には，送信側プロセッサの論理時刻と受信側プロセッサの論理時刻はほぼ同じ値となるため，メッセージの送信時に受信側プロセッサが進みすぎて矛盾となる場合と受信側プロセッサが遅れており矛盾とならない場合が等確率で発生すると考えられるため，矛盾となる確率はほぼ0．5とみなせる．

以上では，矛盾の発生時のキャンセル処理に要する時間は0と仮定して解析を行ったが，次にキャンセル処理のオーバヘッドによる影響を定性的に検討する．

受信側プロセッサで矛盾が発生すると，受信側プロセッサでは処理される有効なイベントの数には変化がないが，無効となるイベントが存在することになり有効イベント当りに必要な実時間が増加することになる．そこで，キャンセル処理1回当りのキャンセル処理に要する実時間を$Tc = k\,Rr$（k：定数）とする．有効イベント1個当りの実時間は矛盾状態発生率PcとTcを用いて

$$Rr + Pc\,Tc = Rr + Pc\,k\,Rr = (1 + Pc\,k)\,Rr \qquad (5\text{-}11)$$

となり，実時間更新量比は

$$\frac{Rr + Pc\,k\,Rr}{Rs} = (1 + Pc\,k)\,\frac{Rr}{Rs} \qquad (5\text{-}12)$$

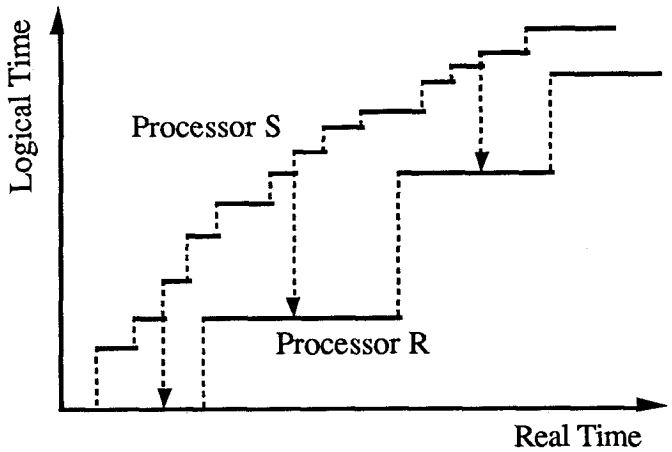


図5-11　プロセッサ間関係
(Rr/Rs) → ∞, (Ls/Lr) → 0

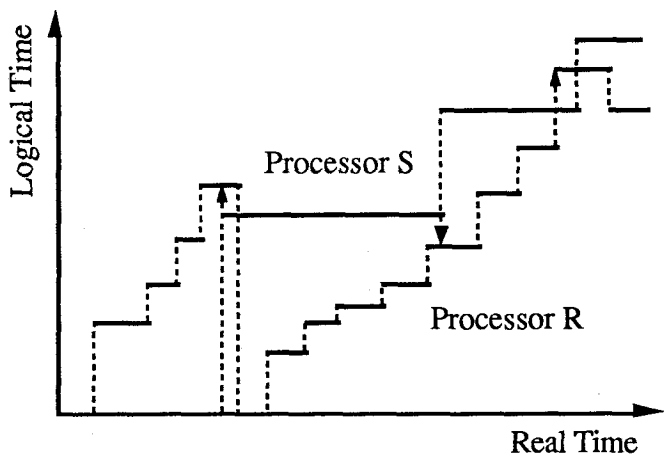


図5-12　プロセッサ間関係
(Rr/Rs) → 0, (Ls/Lr) → ∞

となる．したがって理想処理速度改善比が2となるように，すなわち，式（5-10）の条件を満たすように各プロセッサの実処理速度が等しくなるようにタスクを割当てた場合，キャンセル処理によるオーバヘッドを考慮したときの実時刻更新量比と論理時刻更新量比の関係は図5-13に示すように実時刻更新量比の増加方向にシフト(1)すると考えられる．しかしながら，このようにシフトすると図5-8から分かるように矛盾状態発生率は減少するため，今度は逆方向にシフト(2)すると考えられる．ただし，このシフト(2)はシフト(1)によって矛盾状態発生率が低下するために生じるものであるが，矛盾状態発生率が負となることはないのでシフト(2)のシフト量がシフト(1)のシフト量を越えることはない．したがって，結局これら2つの相反する作用が互いにつりあう所で安定すると考えられる．

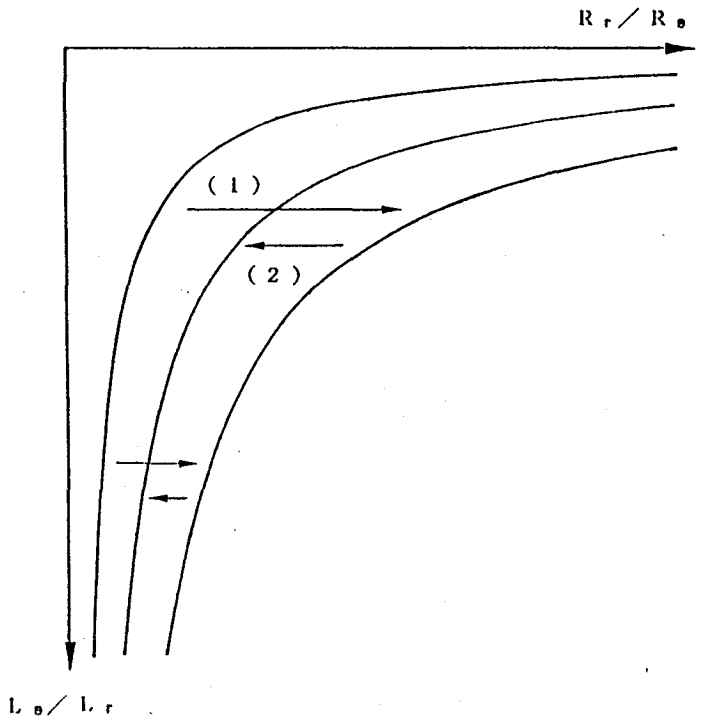


図5-13　キャンセル処理オーバーヘッドの影響

一方，実処理速度改善比は，受信側プロセッサにおける1イベント当りの実時間が（1＋Pc k）倍されることを考慮すると，2／（1＋Pc k）に低下すると考えられる．これが，5．4で述べたdである．しかし，実処理速度改善比の低下はPcが0に近いほど少なく，改善比は2に近づく．

上記の考察から，理想処理速度改善比が2となる，すなわち送信側プロセッサと受信側プロセッサの処理速度が等しくなるように式（5-10）を満たし，なおかつ，矛盾状態発生率Pcが小さくなるように，（Rr／Rs）を大きく，（Ls／Lr）を小さくするように割当てることで実処理速度改善比を2に近づけることができる．具体的には送信側には比較的大きさの小さいタスクを多く，受信側には大きいタスクを少なく，また各プロセッサの負荷が均一となるように割当てることで実現できる．

5.6　結言

本章では拡張メッセージ・パッシングの提案を行った．この拡張メッセージ・パッシングを用いると，メッセージの送受信が確率的な場合や，送受信時の受信側プロセッサの論理時刻が分布している場合のメッセージ送信確率と論理時刻の分布を数学的に取り扱うことができる．

さらに，拡張メッセージ・パッシングによるモデルを用いて処理過程に一時的な矛盾を許容するプロセッサ間同期法である先行制御方式の理論解析を行った．その結果，処理速度低下の主な原因となる矛盾状態の発生率が実時刻更新量比と論理時刻更新量比によって，一意に決定されることを示した．また，理想処理速度改善比が2の条件下では実時刻更新量比を大きく，論理時刻更新量比を小さくする，具体的には負荷が均一になるように送信側プロセッサに比較的小規模のタスクを多く，受信側プロセッサには大規模のタスクを少なく割当てることで矛盾状態発生率を小さくできることを示した．さらに，キャンセル処理によるオーバヘッドが処理速度改善比に与える影響を定性的に検討し，矛盾解消のためのキャンセル処理によるオーバヘッドも考慮した．実処理速度改善比も，送信側に小規模なジョブを多く，受信側に大規模なジョブを少なく

割当てて負荷を均一化するることで改善できることを示した．

第6章

並列処理システムの応用

6.1　緒言

分散データベースシステム(Distributed Data Base Management System : DDBMS)は地理的に分散したデータベースをネットワークを介して物理的に接続し，ユーザからはあたかも1つのデータベースのように機能するシステムである．分散データベースでは複数の処理単位（トランザクション）が分散して同時に共有データにアクセスするため，競合する操作間で同期をとり，データベースの首尾一貫性を保持するための同時実行制御が必要である．従来の同時実行制御方式に時刻印方式がある．時刻印方式は，ユーザからトランザクションを要求されると受付順に時刻印を付与する．またデータ項目毎に，読み出し時刻印，書き込み時刻印を保存しておき，操作衝突時にこれらの時刻印に基づきデータへのアクセス制御を行う．

従来の時刻印方式では，競合する操作間に時刻矛盾が生じたときには，常に小さい時刻印を持つトランザクションがアボートされ，時刻印の付け替えが行われる．したがって，全トランザクションをユーザからの要求順に処理することが保証できず，公平性が損なわれる．また，処理時間の長いトランザクションが終了（コミット）できないロックアウト状態が発生する危険性すらある．

本章では，トランザクションに付与された時刻印を変更することなく，矛盾発生時には時刻印の大きいトランザクションを後退復帰させる不変

時刻印方式[FURU 90a][FURU 90b][KOBA1 91b]の提案を行う．本方式では，コミットメント制御に仮コミットと本コミットの２つのフェーズを設けている．トランザクションは処理の終了時に仮コミットされ，後退復帰の可能性がなくなると本コミットされユーザに結果が返される．本コミット許可の決定には，システム内で処理中のトランザクションの時刻印の最小値であるグローバル処理最小時刻印（ＧＴＡ）が用いられる．ＧＴＡは各ノード間でトークンを巡回することで決定することができる．

また，本章で提案した不変時刻印方式が直列可能性を満たすと同時に，デッドロックやロックアウトが生じない方式であることを証明する．すなわち本方式は，受付順にトランザクションが処理されるため公平性が損なわれれず．しかも処理時間の長いトランザクションに対してもロックアウトが発生しないという特徴がある．

6.2　分散データベースへの応用

分散データベースにおいて，ネットワークに結合されたローカルなデータベースシステムをサイトとよび，ユーザから要求される１つのまとまった処理単位をトランザクションと呼ぶ．トランザクションはいくつかの読み出しステップ(read step)と書き込みステップ(write step)からなる系列である．なお，本論ではトランザクションＴiのデータ項目ｘへの読み出しステップをＲi(x)，書き込みステップをWi(x)と表すことにする．

分散データベースでは，処理装置のアイドル時間をできるだけ少なくし処理効率を向上させるために，複数のトランザクションが分散して同時に共有データへのアクセスを行う．このため，複数のトランザクションによる同一データへのアクセスが衝突すると，この衝突操作の実行順序により，個々のトランザクションは正しい操作を行っていたとしても，データベースシステム全体としてはデータ間で満たさなければならない首尾一貫性(consistency)が犯される可能性がある．たとえば列車の予約システムを考えると，予約済みの座席数と残っている座席数の和は一定でなければならない．予約済みの座席数ｒの更新を行う次のようなトランザクションＴresを考える．

```
Tres:   R(r)
        r := r + 1
        W(r)
```

R(r)はデータ項目rの読み出しステップ，W(r)は書き込みステップであり，このトランザクションには，データ項目rへの読み出し，書き込みステップが2つ含まれており，データ項目へのアクセスを行う読み出し，書き込み操作が衝突を引き起こす．以後トランザクションの操作を，衝突により，首尾一貫性に影響を及ぼす読み出し，書き込みステップのみに注目し議論する．

今，A，Bという2人の客がほぼ同時に座席の予約を行い，上のトランザクションTresを実行したとすると，Tres-A，Tres-Bに含まれる読み出しRA(r)，RB(r)，書き込みWA(r)，WB(r)の合計4ステップの実行系列は表6-1のように6通り考えられる．表6-1に示されるようなトランザクション集合Tの全ステップからなる系列を，Tのスケジュールと呼ぶ．データ項目rの初期値を10とすると，2人の客によって座席が予約された後のrの値は12になっていなければならない．トランザクションTres-A，Tres-Bを表6-1のスケジュール1，2で実行するとrの最終値は12となるが，スケジュール3，4，5，6で実行するとrの最終値は11となってしまう．このような状態を首尾一貫性が犯されたという．

DDBMSにおいて，データベースの首尾一貫性を保証するために，複数のトランザクションの処理の制御を行う機構が同時実行制御である．

表6-1　実行系列

1	RA(r)	WA(r)	RB(r)	WB(r)
2	RB(r)	WB(r)	RA(r)	WA(r)
3	RA(r)	RB(r)	WA(r)	WB(r)
4	RA(r)	RB(r)	WB(r)	WA(r)
5	RB(r)	RA(r)	WA(r)	WB(r)
6	RB(r)	RA(r)	WB(r)	WA(r)

6.3　分散データベースの同時実行制御

トランザクションを小さく切り分けて実行した場合，共有データに対するアクセスの順序により，さまざまな処理結果が得られる．これらの処理結果の内，各トランザクションを１つずつ直列に処理していく場合には正しい結果が得られると考えられる．したがって，トランザクションを並列に処理した結果が，直列に処理した場合の結果と等しければ，並列処理スケジュールは正しい結果を得ていると判断できる．このように，あるスケジュールがトランザクションを直列に実行した結果と等しければ，このスケジュールを直列可能なスケジュールと呼び，このときデータベースの首尾一貫性が守られるという．本節では，直列可能性について定義し，直列可能性を満たす同時実行制御方式である時刻印方式について述べる．

6．3．1　分散データベースの正当性

分散データベースの同時実行制御の妥当性を判定する概念に直列可能性(serializability)がある．本研究では直列可能性を以下のように定義する．前節で述べたようにトランザクション集合Ｔ＝｛Ｔ1，Ｔ2，…，Ｔn｝のスケジュールとは，Ｔの全ステップを実行順に左から右に並べた系列をいう．スケジュールの実行に際しては，形式的な初期トランザクションＴ0＝Ｗ0(Ｄ)と最終トランザクションＴf＝Ｒf(Ｄ)が伴っているものとする．ただし，Ｄはデータベースシステムのデータ項目の有限集合で

$$D = \{d1, d2, \cdots, d|D|\} \qquad (6\text{-}1)$$

からなるものとする．|Ｄ|はＤの位数を表す．すなわち，Ｔ0は全データ項目の初期値を書き込み，Ｔfは全データ項目の最終値を読み出すトランザクションである．

直列なスケジュールとは，各トランザクションの読み出し，書き込みステップをトランザクション毎にまとめて順番に並べた系列をいう．たとえば，Ｄ＝｛ｘ，ｙ｝，Ｔ1：Ｒ1(x)Ｗ1(y)，Ｔ2：Ｒ2(x)Ｗ2(x,y)，Ｔ3：Ｒ3(x,y)Ｗ3(y)のとき，Ｔ1Ｔ2Ｔ3の順に並べた直列スケジュールＬsは次のようになる．

$$Ls = W0(x,y)R1(x)W1(y)R2(x)W2(x,y)R3(x,y)W3(y)Rf(x,y) \quad (6-2)$$

これ以外にも，T1T3T2やT2T1T3など合わせて6通りの直列スケジュールが存在する．

スケジュールLの各読み出し操作Ri(x)に対して，データ項目xへの書き込み操作でRi(x)よりも以前に行われるものの中で最新のものがWj(x)であるとき，スケジュールLにはデータ項目xについてTiとTjにRF関係があるといい，

$$(x, Ti, Tj) \in RF(L) \quad (6-3)$$

と記し，TiはTjが書き込んだxの値を読み出すという．ある書き込み操作Wi(x)が更新したxの値を読み出す操作が存在しないとき，Wi(x)は無効書き込み操作であるという．

Tに対する2つのスケジュールL1とL2において，すべての読み出し操作Ri(x)に対して

$$(x, Ti, Tj) \in RF(L1) \quad (6-4)$$

かつ

$$(x, Ti, Tj) \in RF(L2) \quad (6-5)$$

が成り立つとき，スケジュールL1とL2は等価であるという．ある直列スケジュールLsと等価なスケジュールLを直列可能なスケジュールと呼ぶ．

直列可能性を判定するには，各データ項目における競合操作の実行順序関係を何らかの方法で表現し，直列スケジュールの場合と比較して等価であるかどうかを確認しなければならない．直列可能性の判定法にはＤＩＴＳによる方法，石置きゲームによる方法などがある[MURO85]．以下ではＤＩＴＳによる方法[HARA87]について簡単に述べる．

Ｔに対するスケジュールＬについてＴＩＯグラフ(Transaction Input/Output Graph)を次のように定義する．ＴＩＯ（Ｌ）の節点はＴ内のトランザクション（Ｔ0，Ｔfを含む）に対応し，（ｘ，Ｔi，Ｔj）∈ＲＦ

（Ｌ）ならば，ラベルｘを持つ有向枝（Ｔi，Ｔj）：ｘを付す．Ｔiが無効書き込みWi(x)を含むとき，ダミー節点Ｔi'を追加し，追加枝（Ｔi，Ｔi'）：ｘを付す．ある節点から発し，同じラベルを持つ枝の集合をインターバル(interval)という．

ＴＩＯ（Ｌ）の節点間に次の２つの性質を持つ全順序を定めることができるとき，ＴＩＯ（Ｌ）はＤＩＴＳ(Disjoint Interval Topological Sort)を持つという．

i) Ｔi≪ＴjならばＴjからＴiへの有向路は存在しない．
ii) 排除規則：ｇ≠ｉを満たす有向路（Ｔg，Ｔh）：ｘ，
（Ｔi，Ｔj）：ｘに対し，Ｔg≪ＴjならばＴh≪Ｔiが成立する．

図６-１にＴＩＯグラフの例を示す．図６-２に図６-１のＴＩＯグラフ中のインターバルを示す．このＴＩＯグラフがＤＩＴＳを持つかどうかを調べると，図６-３のような結果となりラベルａを持つ２つのインター

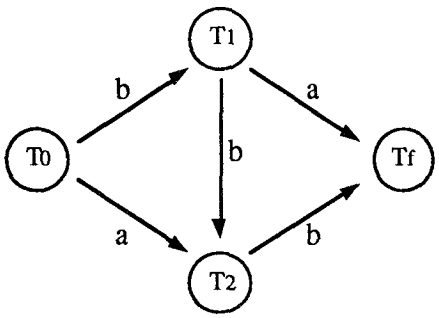


図6-1　ＴＩＯグラフ

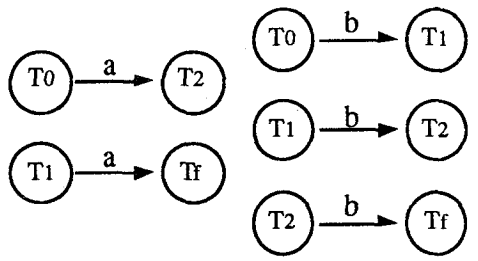


図6-2　インターバル

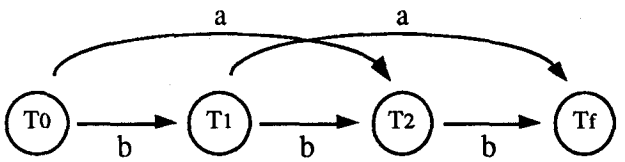


図6-3　ＤＩＴＳの判定

バル（T0, T2）：aと（T1, T4）：aがオーバーラップしており，上記のii）を満たすことができず，ＤＩＴＳが存在しないことがわかる．すなわち，有向路を持たないＴＩＯ（Ｌ）に対し，全節点をT0を最初としTfが最後となるように左から右に枝方向に矛盾しないように一列に並べたとき，同じラベルを持つ２つ以上のインターバルが重ならないことが，ＤＩＴＳの存在する条件である．以上の定義のもとで次の定理が成立する．

（定理）
スケジュールＬが直列可能であることの必要十分条件はＴＩＯ（Ｌ）がT0を最初としてTfを最後とするＤＩＴＳを持つことである．

次のような（6-2）と等価なスケジュールＬについて考える．

$$L = W0(x,y)R1(x)R2(x)W2(x,y)R3(x,y)W1(y)W3(y)Rf(x,y) \qquad (6-6)$$

スケジュールＬのＴＩＯグラフは図6-4のようになる．またＬのインターバルを図6-5，ＤＩＴＳを図6-6に示す．ＴＩＯ（Ｌ）にはＤＩＴＳが存在しているためスケジュールＬは直列可能なスケジュールであると判定できる．

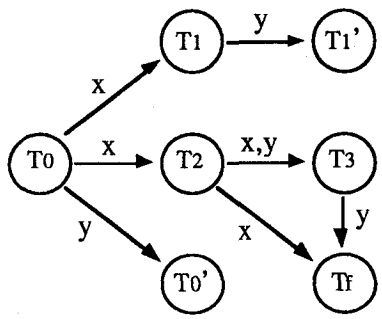


図6-4　ＴＩＯ（Ｌ）

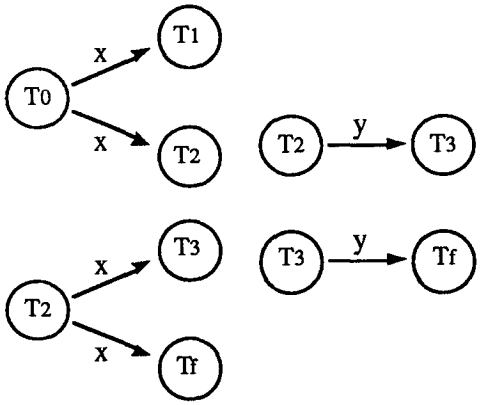


図6-5　インターバル

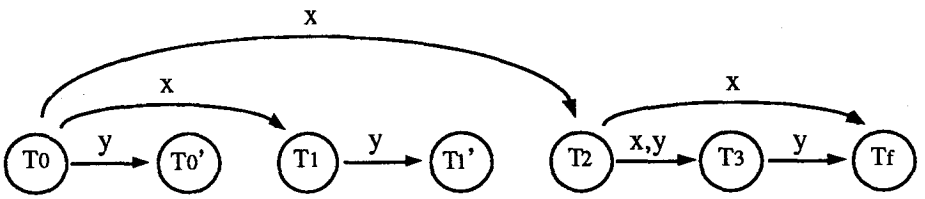


図6-6　DITS

6.3.2　時刻印方式

分散データベース内では，複数のトランザクションが同時に処理実行されるため，データの首尾一貫性を保持するためのトランザクションの同時実行制御が必要となる．従来から提案されてきた同時実行制御方式の1つに時刻印方式がある．

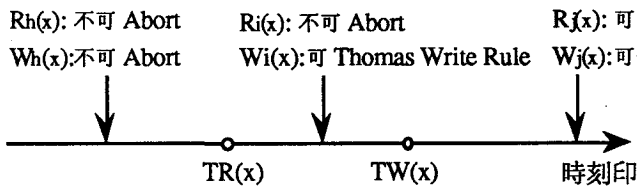
時刻印方式は各トランザクションに要求受付順にシステム全体で唯一の時刻印を付与し，同一データへの操作競合が発生した場合には時刻印の大小でトランザクションの実行を制御し，直列可能性を満足させる方式である．時刻印方式は以下に示すように基本時刻印方式と多版時刻印方式の2つに大別できる．

(i) 基本時刻印方式

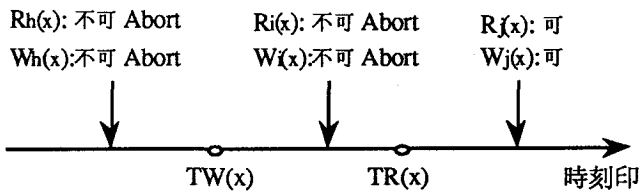
トランザクションTiの時刻印をTSiとする．各データ項目xには，データ項目xを読み出したトランザクションの時刻印の最大値である読み出し時刻印TR(x)とデータ項目xに書き込みを行ったトランザクションの時刻印の最大値である書き込み時刻印TW(x)が記録されている．

Tiによってデータ項目xの読み出し操作が要求された場合には，TSiとTW(x)を比較し，TSi>TW(x)ならば読み出し可であり現在の値を読み出し，TR(x)をTR(x)とTSiのうち大きい方の値に更新する．それ以外の場合は，Tiをアボートし時刻印を付け替え，トランザクションを再処理する．

Tiによってデータ項目xの書き込み操作が要求された場合には，TSiとTR(x)，TW(x)を比較し，TSi≧max（TR(x)，TW(x)）ならば書き込み可であり書き込みを実行し，TW(x)をTSiに更新する．また，TR(x)<TSi<TW(x)ならば書き込みを無視し，Tiは処理を続行することができる（Thomas 書き込み則）．すなわち，Tiによる書き込み操



(a) TR(x)<TW(x)の場合



(b) TW(x)<TR(x)の場合

図6-7 基本時刻印方式

作Wi(x)は無効書き込みとなる．それ以外の場合は，Tiをアボートし時刻印を付け替え再処理する．

図6-7に基本時刻印方式のデータアクセスの許可動作を示す．

(ii) 多版時刻印方式

データ項目xの書き込み操作を実行する毎に，旧版を保存し新たな版を作成する点が基本時刻印方式と異なる．また各版kには，版kの値を読み出したトランザクションの時刻印の最大値である読み出し時刻印TRk(x)，版kを生成したトランザクションの時刻印である書き込み時刻印TWk(x)が付随する．

Tiによってデータ項目xの読み出し操作が要求された場合には，TSi>TWk(x)を満たす最大の書き込み時刻印を持つ版kを選択し，読み出しを実行する．また，TRk(x)をTRk(x)とTSiのうち大きい方の値に更新する．

Tiによってデータ項目xの書き込み操作が要求された場合には，まずTSiと最新版nのTRn(x)，TWn(x)を比較し，TSi≧max（TRn(x)，TWn(x)）ならば書き込みと新たな版（n＋1）の作成を行い，書き込み時刻印TWn+1(x)をTSiとする．TSi<TWn(x)のときには，TSi>TWm(x)を満たす最大の書き込み時刻印を持つ版mを選択し，TSi≧TRm(x)ならばデータ項目xへの書き込みは無視し，新たな版jを作成し挿入する（Thomas 書き込み則）．版jの書き込み時刻印TWj(x)をTSiとする．それ以外の場合は，Tiをアボートする．

図6-8に多版時刻印方式のデータアクセスの許可動作を示す．

多版時刻印方式の特徴として，読み出し操作時にアボートが発生しないという点が挙げられる．

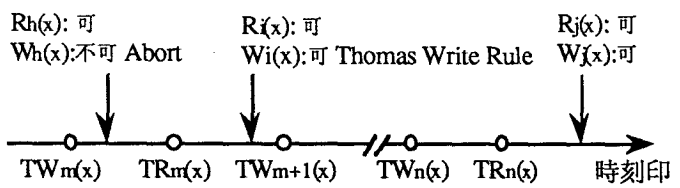


図6-8 多版時刻印方式

6.4　不変時刻印方式

従来の時刻印方式では，操作衝突時には先に実行を開始した時刻印の小さいトランザクションをアボートするために，要求順にトランザクションを処理することができず，公平性が損なわれる．また，トランザクションの到着はマルコフ到着過程と考えられるため，処理時間の長いトランザクションは処理中に他のトランザクションが到着する確率が高く，繰り返しアボートされコミットできないロックアウト状態発生の可能性すらある．そこで，これらの問題を解決する時刻印方式として，不変時刻印方式の提案を行う．

以下に不変時刻印方式の基本動作について述べる．なお，以下では，

ＴＳｉ　：トランザクションＴｉの時刻印

ＴＲｋ（ｘ）：データ項目ｘの版ｋの読み出し時刻印

ＴＷｋ（ｘ）：データ項目ｘの版ｋの書き込み時刻印

と定義する．また，各トランザクションの時刻印は，受付サイトのローカルクロックを用いて行う．また，時刻印は，トランザクション受付時の実時刻とサイト識別番号を組み合わせることで，同一の時刻印を持つトランザクションが存在しないことを保証する．

また，版ｋには，版ｋに対して行った全ての操作の履歴が保存されている．（図6-9）

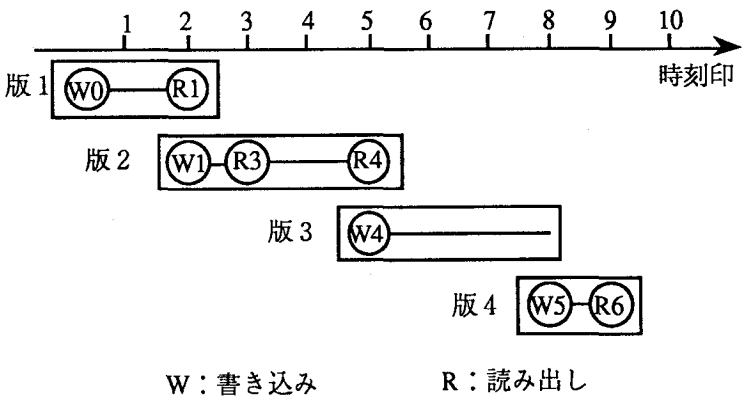


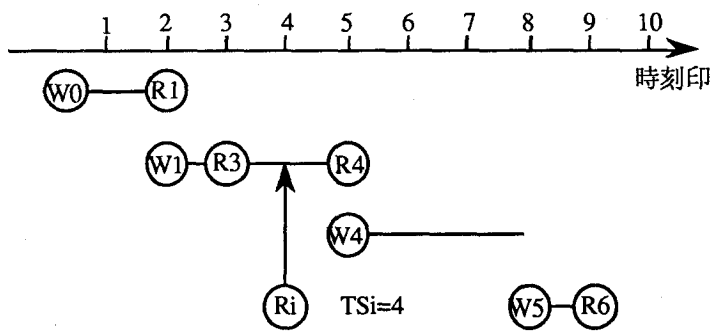
図6-9 不変時刻印方式の版管理

以下に不変時刻印方式の操作の詳細について述べる．

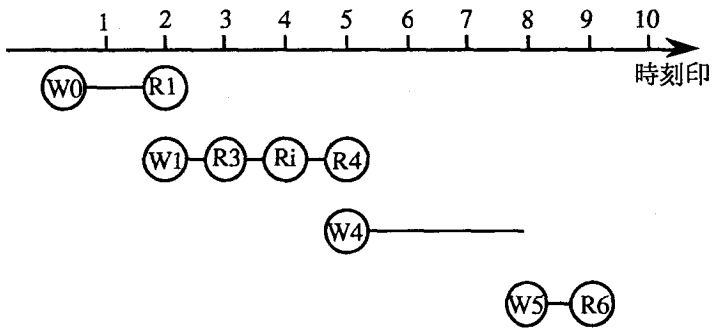
1）　読み出し操作

不変時刻印方式の読み出し操作は，多版時刻印方式の読み出し操作と同様にアボートされることなく実行可能である．

具体的には，トランザクションTiによる読み出し操作Ri（x）が要求された場合には，TSi>TWk（x）を満たす最大の書き込み時刻印を持つ版kを選択し，読み出しを実行し，TRk（x）をTRk（x）とTSiの大きい方の値に更新する．また，処理の後退復帰を実現するために，各版には読み出し操作を行ったトランザクションの起源サイト，



(a) 読み出し実行前のデータの履歴



(b) 読み出し実行後のデータの履歴

図6-10　読み出し

時刻印の値をデータの履歴に記録しておく．たとえば，図6-10では読み込み操作Riの時刻印は4である．このとき，書き込み操作W1，W4の時刻印はそれぞれ2と5であるため，読み込み操作Riは書き込み操作W1が作成した版から読み出しを行い，読み出し操作を行ったことをその版の履歴に記録しておく．

2）　書き込み操作

トランザクションTiによる書き込み操作Wi(x)が要求された場合には，次のような手順で制御を行う．

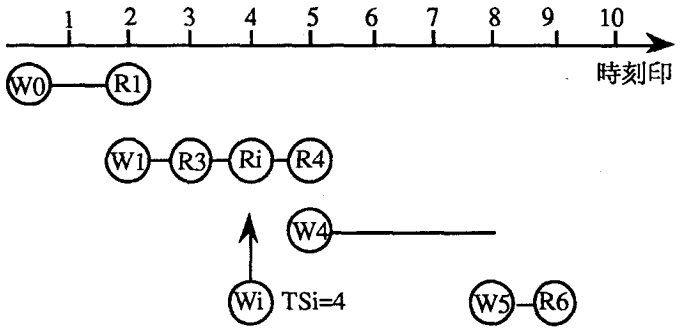
（ステップ1）TSi>TWk(x)を満たす最大の書き込み時刻印を持つ版kを選択する．

（ステップ2）a) TSi≧TRk(x)ならば無矛盾であり，新版jを作成しTWj(x)をTSiとする．

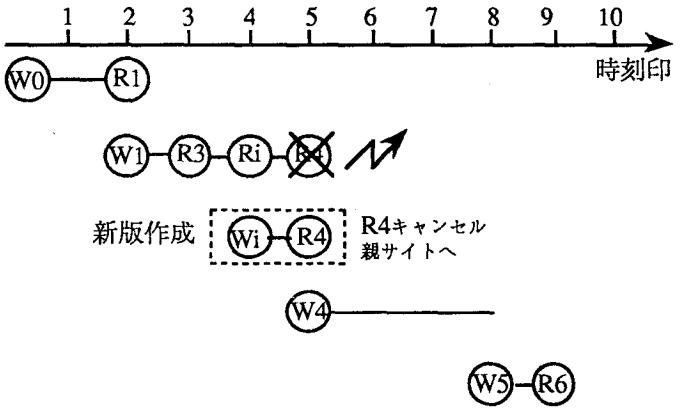
b) TSi<TRk(x)ならば時刻矛盾が発生する．
まず，版kの処理履歴からTSiより大きな時刻印を持つ実行済み読み出し操作をキャンセルする．次に新版jを作成し，TWj(x)をTSiとする．さらにキャンセルされた読み出し操作を再実行し，Wi(x)が書き込んだ値を読み出し，結果を親サイトに返す．

図6-11に矛盾が発生した場合の書き込み操作実行前後のデータの履歴を示す．この図の場合には，時刻印4を持つ書き込み操作Wi(x)と時刻印5を持つ実行済み読み出し操作R4(x)との間に時刻矛盾が発生している．このため，上記の手順にしたがってR4(x)をキャンセルし，Wi(x)による新版作成後，R4(x)を再実行している．R4(x)の再読み出しの結果は親サイトに返される．

このように，不変時刻印方式は書き込み操作が矛盾を引き起こす場合に，トランザクションがアボートされることがなく，読み出し操作のキャンセル・再処理によって矛盾を解消することができる（すなわち，常に書き込み可である）．



(a) 書き込み実行前のデータの履歴



(b) 書き込み 実行後のデータの履歴

図6-11　書き込み

3）　後退復帰

矛盾発生により読み出し操作をキャンセルされたトランザクションの親サイトでは，再読み出し実行の通知を受け取るとトランザクションの処理系列をキャンセルされた読み出し操作まで後退復帰する．その後，再処理を行い，実行済み書き込み操作の値が変更された場合には，書き込み操作のキャンセル要求を出し再書き込みを行う．

トランザクションTiによって実行済み書き込み操作Wi(x)の再書き込みが要求されると，データ項目xでは処理履歴からWi(x)が作成した版を選択し，この版の履歴に実行済み読み出し操作があればこれをキャンセルし，Wi(x)の再書き込み実行後，再読み出しを行う．

このように，再書き込み要求は新たに読み出し操作のキャンセルを伴う可能性があり，後退復帰の連鎖が起こりうる．しかしながら，この連鎖は有限であることを後ほど証明する．

4）　コミットメント制御

不変時刻印方式ではトランザクションの実行終了後も矛盾発生によって後退復帰される可能性がある．したがって，トランザクション終了後直ちに結果をユーザに返すことはできない．ユーザに結果を返すことができるのは，そのトランザクションが後退復帰されないことが保証された後である．

そこで，トランザクションのコミットメント制御に仮コミットと本コミットという2つのフェーズを設ける．仮コミット状態とは，トランザクション終了後，後退復帰の可能性があるため，後退復帰されないことが保証されるのを待っている状態である．本コミット状態とは，トランザクションの後退復帰の可能性がなくなり，ユーザに結果を出力できる状態である．

矛盾が発生するのは，あるトランザクションが読み出し操作実行後，それよりも小さい時刻印を持つ書き込み操作が要求されたときである．したがって，実行中（アクティブ）のトランザクションの時刻印の最小値よりも小さい時刻印を持つトランザクションは，後退復帰されることはない．また，読み出しステップのみからなるトランザクションは，他のトランザクションの後退復帰を引き起こすことがない．

これらのことを考慮して，ローカル処理最小時刻印ＬＴＡ(Local minimum Timestamp of Active transactions)，グローバル処理最小時刻印ＧＴＡ(Grobal minimum Timestamp of Active transactions)を次のように定義する．

ＬＴＡn　：サイトｎで処理中の書き込みステップを含む（あるいは含む可能性のある）トランザクションの時刻印の最小値

ＧＴＡ　：全サイトのＬＴＡの最小値

ＧＴＡは全システムで処理中の書き込みステップを含む（あるいは含む可能性のある）トランザクションの時刻印の最小値となる．すなわち，ＧＴＡより小さい時刻印を持つトランザクションは後退復帰される可能性がないことが保証される．

各サイトｎはトランザクションＴｉを終了すると仮コミットし，ＬＴＡｎを次のように更新する．

ＬＴＡｎ := min(Ｔｓｋ | Ｔｋは書き込みステップを含む
実行中のトランザクション)　（6-7）

また，仮コミット済みのトランザクションＴｉが後退復帰されると

ＬＴＡｎ := min（ＬＴＡｎ, ＴＳｉ)　（6-8）

とした後，Ｔｉを再処理する．

各サイトは何らかの方法でＧＴＡの値を決定し，ＧＴＡより小さい時刻印を持つトランザクションを本コミットし，結果をユーザに返す．

図6-12の例では，ＧＴＡは9であり，各ノードにおいて時刻印が9以前である操作は本コミット済みとなっている．また，時刻印がＧＴＡとＬＴＡの間である操作は仮コミット済みとなっている．たとえば，ノード1では，時刻印7の読み出し操作と時刻印9の書き込み操作は本コミット済みであり，ＬＴＡが13であるため時刻印11の読み出し操作と時刻印13の書き込み操作は仮コミット済みとなっている．

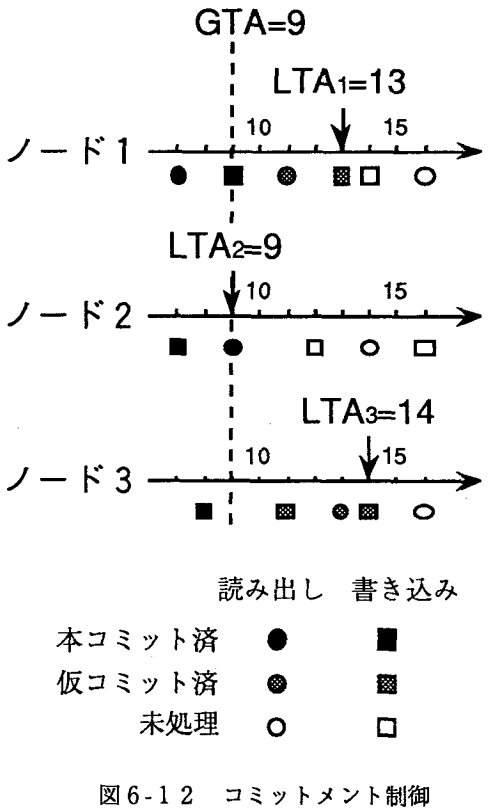


図6-12　コミットメント制御

5）　グローバル処理最小時刻印の決定法

仮コミット済みトランザクションの時刻印がＧＴＡより小さくなれば，そのトランザクションは後退復帰されないことが保証される．本節ではＧＴＡを決定する手法について述べる．

ＧＴＡ決定法は以下のように集中管理と分散管理に大別される．

・集中管理

中央に管理用プロセッサを設ける．たとえば，各サイトがＬＴＡを更新する毎に中央の管理用プロセッサに知らせ，管理プロセッサがそれらの最小値を求めＧＴＡとする方法が考えられる．

・分散管理

管理プロセッサを設けず，各サイトでＧＴＡを決定する．たとえば，全サイトのＬＴＡを記録したトークンをサイト間に巡回させ，トークンを受け取ったサイトは自サイトのＬＴＡを更新し，トークンに書かれている全サイトのＬＴＡの最小値をＧＴＡと決定する方法が考えられる．

集中管理では，システムが大規模になると中央の管理プロセッサへの通信量や処理量が大きくなり，高速化の障害となると考えられる．そこで以下ではトークンを用いた分散管理について考察する．

トークンには全サイトのＬＴＡの値を記録し巡回させるだけでは不十分である．これはトークン巡回の時間差によって誤ったＧＴＡを検出するサイトが発生する可能性があるためである．ここで，

ＧＴＡn：サイトｎがトークンから算出したＧＴＡ決定値
（≦ＧＴＡ）

とする．

たとえば図6-13に示すようにサイトＡ，Ｂ，ＣからなるＤＤＢＭＳにおいて，ＬＴＡA＝８，ＬＴＡB＝１３，ＬＴＡC＝１１のとき，サイトＡからデータｂ（∈Ｂ）への書き込み要求Ｗi(b)が到着したとする．このときサイトＣを起源とする読み出し操作Ｒj(b)は既処理であるとする．この結果，データｂにおいて矛盾が発生し，Ｒj(b)がキャンセルされ仮コ

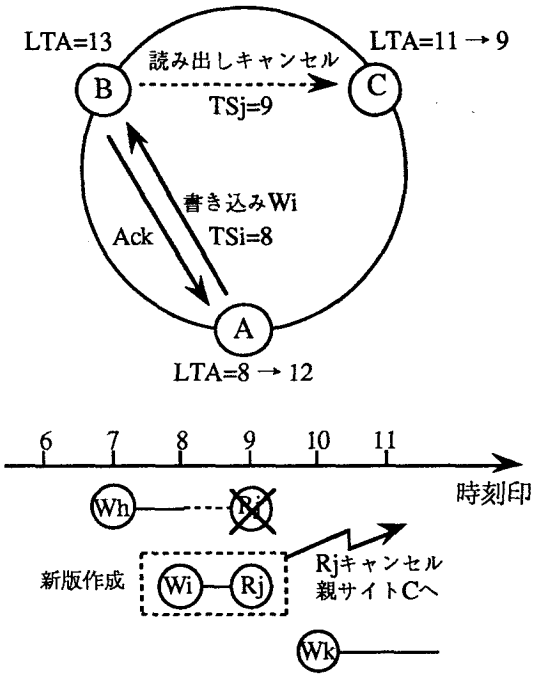


図６-１３　キャンセルの発生例

ミット済みトランザクションＴjは後退復帰されアクティブとなる．Wi(b)の完了後Ａｃｋが親サイトＡに返されるとＬＴＡA＝１２，ＬＴＡC＝９となるが，更新したＬＴＡの値をトークンに反映させるまでの時間差が問題となり，図６-１４（ａ）に示すような状況では次にトークンを受け取るサイトＢはＧＴＡB＝１１と誤って認識してしまう．

そこで，トークンをＬＴＡ部とキャンセル宣言部からなるものとし，キャンセル発生サイト（Ｂ）はキャンセルを誘引した書き込み操作が完了したことを知らせるＡｃｋに，キャンセルさせた読み出し操作の親サイト（Ｃ）と時刻印（９）の情報を載せる．このＡｃｋを受け取ったサイトＡは，トークンが到着すると自サイトのＬＴＡ部を１２に更新し，サイトＣのキャンセル宣言部に９を登録する（図６-１４（ｂ））．

全サイトのＬＴＡ部とキャンセル宣言部を持つＧＴＡ決定トークンを巡回させる場合の，各サイトｎのＧＴＡnの決定手順は次のようにな

る．

1）各サイトｎはトークンを受け取ると，自サイトのＬＴＡ部を更新し，キャンセルを誘発した場合は，該当するサイトのキャンセル宣言部に登録する．
2）読み出し操作のキャンセル・後退復帰を受けたトランザクションが自サイトに存在すれば，トークンのキャンセル宣言部から該当する時刻印を削除する．
3）トークンに記録された全サイトのＬＴＡとキャンセル宣言の最小値を算出しＧＴＡnとする．

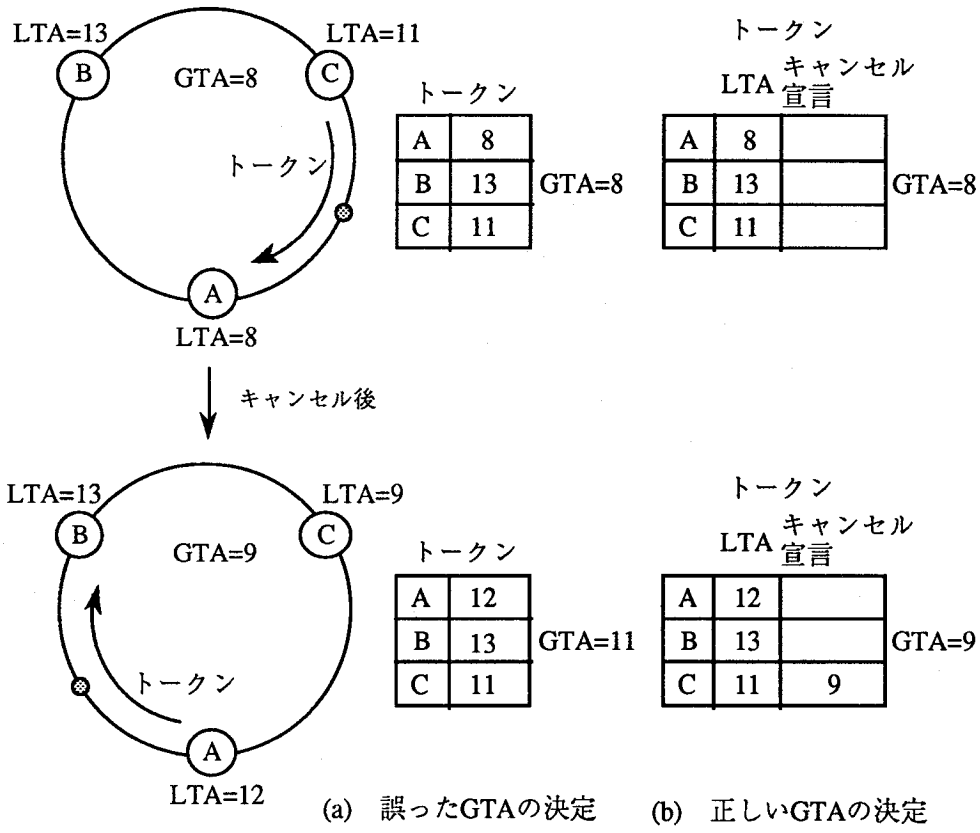


図6-14　トークンの構成

6.5　不変時刻印方式の正当性証明

本節では，不変時刻印方式の正当性を証明する．まず，不変時刻印方式が直列可能性[ESWA 76]を満たすことを示し，次にデッドロック，ロックアウトが生じないことを示す．また，ＧＴＡ決定トークンには各サイトのＬＴＡとキャンセル宣言があれば，ＧＴＡを正しく決定できることを証明する．

1）直列可能性

不変時刻印方式で実行した任意のスケジュールの結果が，トランザクションを時刻印順に並べた直列スケジュールの実行結果と等価であることを示す．

（性質1）

不変時刻印方式では時刻印の順序でＲＦ関係をなすスケジュールが得られる．すなわち，トランザクションＴiがデータ項目ｘの読み出し操作Ｒi(x)を含むとき，Ｒi(x)が読み込む値はＴＳk＜ＴＳi，Ｗk(x)∈Ｔkなるトランザクションのうち最大の時刻印を持つトランザクションＴkの書き込みステップＷk(x)が書き込んだ値である．したがって

（ｘ，Ｔk，Ｔi）∈ＲＦ（Ｌ）

が成立する．

■

（証明）

（ｘ，Ｔk，Ｔi)∈ＲＦ(Ｌ)であり，かつＴＳk＜ＴＳj＜ＴＳi，Ｗj(x)∈Ｔjなるトランザクション Ｔjが存在すると仮定する．これは，ＴiがＴkからｘの値を読み出した後，Ｔjがｘに書き込みを行った場合である．

このような場合には，Ｔjによる書き込み操作Ｗj(x)は図6-11に示したように，データｘにおいて新版を作成しＲi(x)をキャンセルし，Ｒi(x)にＷj(x)が書き込んだ値を新たに読み出させる．したがって

（ｘ，Ｔk，Ｔi）∉ＲＦ（Ｌ）

（ｘ，Ｔj，Ｔi）∈ＲＦ（Ｌ）

が成立する．よって

（x，Tk，Ti）∈ RF（L），
TSk＜TSj＜TSi，Tj(x)∋Wj(x)
なるトランザクションTjは存在しない．

■

（性質1）により不変時刻印方式で実行した結果は時刻印順にトランザクションを並べた直列スケジュールLsとRF関係が同一（RF（L）≡RF（Ls））である．したがって，不変時刻印方式は直列可能性を満たす．

2）デッドロック，ロックアウト
不変時刻印方式ではデッドロック，ロックアウト（特定のトランザクションのコミットが無限に遅らされる）が起こらないことを示す．

（性質2）
トランザクションに与えられた時刻印は，トランザクションがコミットされるまでに変更されることはない．

■

（証明）
不変時刻印方式において，時刻印を変更する手続きは存在しない．

■

（性質3）
グローバル処理最小時刻印GTAが減少することはない．

■

（証明）
GTAが減少するのは，本コミット済みトランザクションが後退復帰されてアクティブとなるときである．しかし，GTAよりも小さい時刻印を持つトランザクションは存在しないので，本コミット済みトランザクションが後退復帰されることはない．よって，GTAが減少することはない．

■

（性質４）
グローバル処理最小時刻印ＧＴＡは必ずいつかは増加する．

■

（証明）
ある時刻にデータベースシステム全体でアクティブなトランザクションのうち最小の時刻印ＴＳkを持つ（ＴＳkはグローバル処理最小時刻印ＧＴＡと一致する）トランザクションをＴkとする．すなわち，ＴＳi＜ＴＳkなる時刻印ＴＳiを持つトランザクションＴiは本コミット済みである．

トランザクションＴkの処理が止まることはないため，ＧＴＡが無限に同じ値であるということは，Ｔkが無限に後退復帰され続けるということを含意する．

1）Ｔkが後退復帰されるのは，Ｔkの読み出し操作Ｒk(x)がキャンセルされたときである．
2）Ｒk(x)がキャンセルされるのは，Ｒk(x)の実行後ＴＳj＜ＴＳkなるＴj（アクティブ）によってＷj(x)が実行されたときである．
3）しかしＴＳj＜ＴＳkなるアクティブなＴjは存在しないので，Ｔjが後退復帰されることはない．
4）したがって，Ｔkは必ず有限時間内に仮コミットし，ＧＴＡは更新されて増加する．

よってＧＴＡは必ずいつかは増加する．

■

（性質２，３，４）により任意のトランザクションＴiの時刻印ＴＳiはいつかはＧＴＡより小さくなり，コミットされる．

よって不変時刻印方式ではデッドロック，ロックアウトは起こらない．

３）トークンによるＧＴＡの決定

前節でＧＴＡ決定トークンがＬＴＡ部とキャンセル宣言部からなることを述べた．もし，トークンの値から決定したＧＴＡnの値が以前の値に戻った（すなわち減少した）とすると，本コミットしてはならないト

ランザクションを本コミットしてしまったことになり，矛盾発生により読み出し操作がキャンセルされたときに後退復帰できなくなる．このような状況が発生すると，実行スケジュールの直列可能性が満たされなくなる．そこで，ＬＴＡ部とキャンセル宣言部からなるトークンを用いたＧＴＡの決定値ＧＴＡnが減少しないことを示す．

（性質５）
トークンから決定されるＧＴＡnの値は減少しない．

■

（証明）
あるサイトｎがトークンのＬＴＡ部を更新した結果，トークンから決定されたＧＴＡnの値が減少し，自サイトｎのＬＴＡnの値になったと仮定する．一つ手前のサイト（ｎ－１）が決定したＧＴＡn-1をＧＴＬ，減少後の決定値ＧＴＡnをＧＴＳ（＜ＧＴＬ）とする．

このサイトｎが以前にトークンを受け取ったときには，時刻印ＧＴＳを持つトランザクションＴsは仮コミット状態であったはずである（もしアクティブならば，このサイトｎのＬＴＡnはＧＴＳ以下となるため，ＧＴＡの決定値ＧＴＡnがＧＴＬ（＞ＧＴＳ）となることはない）．すなわち，サイトｎが前回トークンを更新してから今回更新するまでの間に，時刻印がＧＴＳであるトランザクションＴsが後退復帰されてアクティブになったことになる．これは，ＧＴＳより小さい時刻印ＴＳtを持つトランザクションＴtの書き込み操作が，Ｔsの読み込み操作をキャンセルしたことを含意する．Ｔsの読み込み操作がキャンセルされてアクティブになった時点では，トークンから決定されるＧＴＡの値は，キャンセルを引き起こしたトランザクションＴtの時刻印ＴＳｔ（＜ＧＴＳ）以下であったはずである（トランザクションＴtはアクティブであるから，ＧＴＡ＞ＴＳtとなることはない）．

キャンセルが発生してＴsがアクティブになった後，サイトｎがトークンを受け取ったときの，トークンによるサイト（ｎ－１）が決定したＧＴＡn-1がＧＴＬ（＞ＧＴＳ＞ＴＳt）となっていたということは，このときすでにトランザクションＴtが仮コミットし，Ｔtの親サイトｍのＬＴＡmがＧＴＬ（＞ＴＳt）より大きな値となり，これをトークンに反映

させたことを含意する．しかし，トランザクションＴｔの仮コミット時には，Ｔｔの親サイトｍはトランザクションＴｓの読み出し操作をキャンセルしたことを認識しており，トークンの更新時にサイトｎのキャンセル宣言部にＧＴＳを登録しているはずである．

したがって，サイト（ｎ－１）における決定値ＧＴＡn-1がＧＴＬ（＞ＧＴＳ）となることはない．これはＧＴＡの決定値がＧＴＬからＧＴＳに減少したという仮定に反する．

よってトークンから決定されるＧＴＡの値は減少しない．

■

6.6　結言

分散データベースシステムにおいてトランザクションを受付順に処理できる実行制御方式として不変時刻印方式の提案を行い，その正当性を証明した．

不変時刻印方式は，各データ項目に対して読み出し時刻印及び書き込み時刻印を記録することで，操作衝突時に時刻印の大きいトランザクションの後退復帰を可能としている．また，トランザクションは後退復帰される可能性があるためコミットメント制御に本コミットと仮コミットの２つのフェーズを設けている．トランザクションは処理終了時に仮コミットされ，システム内で処理中のトランザクションの時刻印の最小値であるグローバル処理最小時刻印（ＧＴＡ）がトランザクションの時刻印より大きくなると本コミットされる．

また，不変時刻印方式の正当性の証明として以下の事柄を証明した．まず，不変時刻印方式で実行した結果は，時刻印順にトランザクションを並べた直列スケジュールとＲＦ関係が同一であるため直列可能性を満たすことを証明した．さらに，不変時刻印方式では，トランザクションに与えられた時刻印は変更されることなく，グローバル処理最小時刻印ＧＴＡが減少することなく，いつかは必ず増加するため，デッドロックやロックアウトが生じないことを証明した．

不変時刻印方式では，トランザクションを受付順に処理でき，公平性の高いサービスを実現できる．

第7章

結論

本論文では，利用者に対して並列性記述の負担を負わせることなく，逐次プログラミング言語で記述されたジョブから，最大限の並列性を抽出し並列処理する汎用並列処理システムを対象に，その実現に不可欠な実行制御方式であるジョブ分割法，タスク割当法，およびプロセッサ間同期法について方式の提案ならびに種々の考察を行った．

まず，逐次プログラミング言語で記述されたジョブを並列実行可能なタスクに分割するジョブ分割法としてメッセージ依存分割法と，処理実行時にともに発生することのないタスクを同一のプロセッサに割当てるタスク多重割当法の提案を行った．また，プロセッサ間同期法として従来から提案されてきた先行制御方式を対象にオーバヘッドの理論解析を行った．さらに，並列処理システムの応用として，分散データベースをとりあげ，分散データベースの首尾一貫性を保証する為に必要となる同時実行方式として，ユーザからのトランザクションを受付順にサービスすることのできる不変時刻印方式の提案を行った．

まず第2章では，計算機システムで処理されるジョブを，処理の開始以前に処理過程が決定されているスケジュール可能なジョブと処理の実行にともなって決定されるスケジュール不可能なジョブに分類した．次に，これらのジョブを並列処理する際に必要となる実行制御方式であるジョブ分割法・タスク割当法・プロセッサ間同期法についてその役割について述べた．ジョブ分割法は，並列処理システムに与えられたジョブを並列実行可能なタスクに分割する役割を担っている．ジョブ分割法に

よって得られたタスク集合の持つ並列性を損なうことなく，タスクをプロセッサに割当てるのがタスク割当法である．また，プロセッサ間同期法は，処理結果に矛盾がないようにプロセッサの処理の進行・停止を決定し同期をとるものである．

第3章では，プロセッサ構成単位間の通信に注目しジョブを分割するメッセージ依存分割法の提案を行った．メッセージ依存分割法によって得られるタスクは，メッセージの受信によって起動し，終了時に他のタスクに対してメッセージの送信を行う．このメッセージ依存分割法を従来の分割法と比較した結果，より高い並列性を抽出でき，同程度の並列性を抽出する際には分割回数が少ないことを示した．また，メッセージ依存分割法では，メッセージの送受信点にのみ注目して分割を行うため，従来の分割法では分割できなかったスケジュール不可能なジョブに対しても分割を行うことができることを示した．

第4章では，互いに排反なタスクを同一のプロセッサに割当てることのできるタスク多重割当法の提案を行った．その発生が他のタスクの処理結果によって決定される確率タスクには，実際の処理系列において共に発生することのないタスクが存在する．これらのタスク間の関係をタスクの排反関係という．このような，互いに排反なタスクは，実際の処理系列において共に発生することはないので，同一のプロセッサに割当てても，処理実行時にプロセッサをめぐって競合を起こすことはない．また，同一のプロセッサに割当てると，一方のタスクが発生しなくとも，排反なタスクが発生すれば，プロセッサがアイドル状態とはならない．そこで，タスク間の発生の依存関係である制御依存関係に注目しタスク間の排反性を検出し，互いに排反なタスクを同一のプロセッサに割当てるタスク多重割当法を提案した．この，タスク多重割当法とタスクの排反性を考慮せずに割当てを行うCP/MISF法と比較した結果，プロセッサ台数が限定される場合に，実際に処理を行う際の処理時間が短いことを，また同程度の処理時間を実現するために必要となるプロセッサ台数が少ないことを示した．

第5章では，従来から提案されていたプロセッサ間同期法である先行制御方式を対象に理論解析を行った．先行制御方式では，各プロセッサは互いに他のプロセッサとは独立に処理を行い，矛盾を検出すると進みすぎた処理を取り消し矛盾を解消する方式である．先行制御方式おいて

は，矛盾の発生が処理速度低下の主な原因となっている．そこで，並列処理システムのプロセッサ間の関係をモデル化する概念として拡張メッセージ・パッシングの提案を行い，矛盾状態の発生率を理論解析によって求めた．メッセージ・パッシングは従来からオブジェクト指向におけるオブジェクト間の通信を取り扱う概念として提案されてきたもので，このメッセージ・パッシングに対してメッセージ送受信の有無と送受信時の時刻に確率的要素を導入することで拡張を行った．

解析結果より，２台のプロセッサからなるシステムの場合，送信側に比較的小規模のタスクを多く，受信側に比較的規模の大きなタスクを少なく割当て，負荷を均一にすることで，矛盾状態発生率を０に近づけることができることを示した．また，矛盾解消の為のオーバヘッドの影響を定性的に検討し，処理速度の改善も，負荷が均一となるように送信側に小規模なジョブを多く，受信側に大規模なジョブを少なく割当てることで改善できることを示した．

第６章では，並列処理システムの応用として，分散データベースを取り上げ，分散データベースにおいて重要な概念である首尾一貫性を保証する同時実行制御方式として，トランザクションの到着順サービスを行うことのできる不変時刻印方式の提案を行った．分散データベースでは，ユーザからの処理要求であるトランザクションが各々のサイトで同時処理されるため，データベースの首尾一貫性を保証する同時実行制御方式が必要となる．そこで，データ項目ごとにその書き込みと読み出しの履歴を記録することで矛盾状態の発生時に，時刻印の大きいトランザクションの後退復帰を可能としている．本方式ではトランザクションが後退復帰されるため，トランザクションのコミットメント制御に，仮コミットと本コミットの２つのフェーズを設けている．トランザクションのコミットメント制御は，システム内で処理中のトランザクションの時刻印の最小値であるグローバル処理最小時刻印（ＧＴＡ）をもとに行い，ＧＴＡよりも小さな時刻印のトランザクションのみ本コミットを行いユーザに結果が返される．

また，不変時刻印方式では，時刻印順に処理を行った直列スケジュールと等価な処理結果を得ることができ，デッドロックやロックアウトが生じないことを証明した．

以上，汎用並列処理システム構築に不可欠な実行制御方式を，ジョブ

分割法，タスク割当法およびプロセッサ間同期法に分類し，種々の角度から考察を行ってきたが，効率の並列処理システムを実現するためには今後解決しなければならない問題も多数残されている．それらについて若干述べる．

メッセージ依存分割法では，メッセージの送受信点を分割点とするため，並列性抽出に貢献しない分割を行うことがある．そこで，分割点をいくつかのタイプに分類し，各タイプに属する分割点が並列性抽出に貢献する度合を経験的に学習する能力を持つ，自己学習機能を分割法に持たせる必要がある．

また，タスク割当法として，初期割当ての一種であるタスク多重割当法の提案を行ったが，初期割当てではタスクの処理時間やメッセージの送信の頻度などの正確な値を利用することができず，予測値を用いて割当てを行うため最適な割当てを行うことができない．したがって，処理時間の短縮化を図るためには，適応型割当てを行う必要がある．処理の実行時に，処理対象となるタスク集合の特性を抽出しながら，再割当てを行う適応型割当てでは，どのようなタスク集合の特性を利用するかでその効率が大きく影響をうけると考えられる．今後，適応型割当てを行う際に利用すべきタスク特性を明確にし，収束時間ならびに収束処理効率の高い適応型割当法の提案を行う必要がある．

プロセッサ間同期法として，先行制御方式を用いた並列処理システムでは，矛盾の解消を行うために履歴の保存を行っている．そのため，システムの一部に故障が発生しても，他のプロセッサはこの処理履歴をもとに，故障による処理結果への影響を最小限に押えながら処理を継続できると考えられる．今後，故障がシステムに与える影響を検討し，フォールトトレランス機能を持つ並列処理システム構築について検討を行う必要がある．

謝辞

本研究の全過程を通じ，直接懇切丁寧なる御指導，御鞭撻を賜った大阪大学工学部通信工学科通信網工学講座手塚 慶一教授に心から御礼申し上げる．

また，本論文作成にあたり御助言・御教示を賜った大阪大学産業科学研究所の北橋 忠宏教授に深く感謝する．

学部及び大学院において御指導賜った大阪大学総長熊谷 信昭先生，大阪大学工学部通信工学科の滑川 敏彦名誉教授，中西 義郎名誉教授，倉薗 貞夫教授，森永 規彦教授に厚く御礼申し上げる．

研究を進めるにあたり多大な御教示・御助言を頂いた大阪大学経済学部真田 英彦教授，大阪大学工学部中西 暉助教授，静岡大学工学部渡邊 尚助教授には深謝する．

研究遂行にあたり種々の面でお世話になった九州工業大学情報工学部打浪 清一教授，大阪大学工学部岡田 博美助教授，神戸商船大学井上 健助教授，大阪大学工学部馬場口 登講師，和歌山大学経済学部内尾 文隆講師，大阪大学工学部山本 幹助手，後藤 嘉代子技官には深謝する．

研究の細部にわたり熱心な御討論を頂いた古川 誠氏（現在日本電信電話株式会社勤務），大阪大学大学院村田 英明氏，ならびに小巻 由夫氏（現在キヤノン勤務），大阪大学工学部近藤 博房氏，山本 隆嗣氏には深謝する．

同級生として種々の面でお世話になった大阪大学大学院博士後期課程の金 錫泰氏，大川 剛直氏（現在大阪大学工学部助手），黄瀬 浩一氏（現在大阪府立大学工学部助手）には感謝する．

大学院博士後期課程の戸出 英樹氏をはじめとする大阪大学工学部通信工学科通信網工学講座の諸兄に厚く御礼申し上げる．

参考文献

[ADAM 74] T.L.Adam,K.M.Chandy and J.R.Dickson:"A Comparison of List Schedules for Parallel Processing Systems",Commun.ACM,Vol.17,No.12 (1974)

[AISO 82] 相磯,飯塚,元岡,田中:“計算機アーキテクチャ”,岩波書店(1982)

[AMAN 87] 天野:“マルチプロセッサ型スーパコンピュータ”,電子情報通信学会誌,Vol.70,No.12 (1987)

[BENA 82] M.Ben-Ari : "Principles of Concurrent Programming",Prentice-Hall International(1981) (渡辺:“並行プログラミングの原理”,啓学出版(1986))

[BERN 81] P.A.Bernstein and N.Goodman : "Concurrency Control in Distributed Database Systems",ACM Computing Surveys,vol.13,·No.2 (1981)

[CHAN 78] K.M.Chandy and J.Misra:"A non-trivial example of concurrent processing : Distributed simulation",Pro.COMPSAC,Chicago,Nov.16-18 (1978)

[CHAN 79] K.M.Chandy and J.Misra:"Distributed Simulation:A Case Study in Design and Verification of Distributed Programs",IEEE Trans.Softw.Eng,Vol.SE-5,No.5 (1979)

[CHAN 81] K.M.Chandy and J.Misra:"Asynchronous Distributed Simulation via a Sequence of Parallel Computations",Comm.ACM,Vol.24,No.11 (1981)

[COMP 85] Computer,special issue:"Multiprocessing Technology",IEEE(1985)

[ESWA 76] K.P.Eswaran,J.N.Gray,R.A.Lorie,and I.L.Traiger:"The Notions of Consistency and Predicate Locks in a Database System",Communications of ACM,Vol.19,No.11 (1976)

[FURU 88a] 古川,小林,中西,手塚:“先行制御方式を用いた並列処理システムの性能評価”,電子情報通信学会システムのモデリングと性能評価時限研究専門委員会資料, (1988-6)

[FURU 88b] 古川,小林,中西,手塚:“シミュレーションによる先行制御方式の処理効率解析”,昭和63年電気関係学会関西支部連合大会,S 7-8 (1988-11)

[FURU 90a] 古川,小林,中西,手塚:“要求順サービスを可能とする制御方式について”,電子情報通信学会データベースワークショップ資料, p121-125 (1990-1)

[FURU 90b] 古川,小林,中西,手塚:“要求順サービスを可能とする不変時刻印方式の提案”,1990年電子情報通信学会春季全国大会, D-73 (1990-3)

[GOLD 87] A.Goldberg and D.Robson:"Smalltalk-80:The Language",Addison-Wesley (1987), (相磯訳:“SmallTalk-80 言語詳解”,オーム社(1987))

[GRAH 78] R.L.Graham:“スケジューリングの組み合わせ数学”,別冊サイエンス・コンピュータ数学 (1983)

[HALI 89] U.Halici and A.Dogac:"Concurrency Control in Distributed Databases Through Time Intervals and Short-Term Locks",IEEE Transactions on Software Engineering,Vol.15,No.8 (1989)

[HARA 87] 原嶋,茨木:“先読みスケジューラによる分散型データベースシステムの並行処理制御”,電子情報通信学会論文誌(D),Vol.J70-D,No.6 (1987)

[HEWI 77] C.Hewitt:"Viewing Control Structures as Patterns of Passing Messages",Artificila Interlligence,Vol.8 (1977)

[HOCK 81] R.W.Hockney and C.R.Jesshope : "Parallel Computers : architecture, programming and algorithms",Adam Hilger(1981), (奥川, 黒住共訳:“並列計算機”,共立出版 (1984))

[HWAN 84] K.Hwang and F.A.Briggs:"Computer Architecture and Parallel Processing", McGraw-Hill,New York (1984)

[IBAR 83] 茨木:“組み合わせ最適化”,産業図書 (1983)

[IKED 88] 池田:“分散データベースの同期制御の一方式”,情報処理学会論文誌,Vol.29,No.7 (1988)

[INAM 85] 稲守,戸田:“複数マルチプロセッサを用いた並行型通信網トラヒックシミュレータの評価”,電子通信学会論文誌(B),J68-B,No.1 (1985)

[JEFF 85a] D.Jefferson and H.Sowizral:"Fast concurrent simulation using the time warp mechanism",Pro. of the Conference on Distributed Simulation 1985 (1985)

[JEFF 85b] D.Jefferson:"Implementation of time warp on the Caltech hypercube",Proc. of the Conference on Distributed Simulation 1985 (1985)

[KAMB 83] 上林:“データベースの基礎理論(6)－共有データベースの諸問題に対する理論”,情報処理,Vol.24,No.8 (1983)

[KAMB 86] 上林:“データベース”,昭晃堂(1986)

[KAMB 87a] 上林:“分散データベースとその実現上の問題点”,bit,Vol.20,No.1 (1987)

[KAMB 87b] 上林:“分散処理技術の基本課題”,情報処理,Vol.28,No.4 (1987)

[KAMB 87c] 上林:“可制御二相施錠方式”,情報処理学会研究報告(データベース・システム),87-DB-61-5,Vol.87,No.66 (1987)

[KAMB 87d] Y.Kambayashi and X.Zhong:"Controllable Timestamp Ordering and Oriental Timestamp Ordering Concurrency Control Mechanisms",Proceedings of the IEEE Computer Society's 11th Annual International Computer Software & Applications Conference(COMPSAC) (1987)

[KAMB 88] Y.Kambayashi:"Integration of Different Concurrency Control Mechanisms in Heterogeneous Databases",Proceedings of the second International Symposium on Interoperable Information·Systems(ISIIS '88),ORM Publishing Co. (1988)

[KAMB 89] 上林,最所,仲:“実時間データベースに適した2段階2相施錠方式”,情報処理学会研究報告(データベース・システム)

,89-DBS-72-17,Vol.89,No.63 (1989)

[KASA 84] 笠原,成田:“マルチプロセッサ・スケジューリング問題に対する実用的な最適及び近似アルゴリズム”,電子通信学会論文誌(D),Vol.J67-D,No.7 (1984)

[KASA 88] 笠原:“マルチプロセッサシステムの研究動向”,電気学会論文誌(C),108号 (1988)

[KATA 89] 片岡,武田,佐藤,井上:“多版同時実行制御に関する一考察”,情報処理学会第３９回（平成元年後期）全国大会,5M-6 (1989)

[KAWA 88] 河合,山下,大野,吉村,西村,下條,宮原,大村:“並列画像生成システムLINK-2のアーキテクチャ”,情報処理学会論文誌,29-8 (1988)

[KIKU 88] 菊池,白鳥,宮崎:“逐次型高水準言語プログラムのモジュール分割による並列性の抽出について”,電子情報通信学会論文誌(D),J71-D,No.8 (1988)

[KINI 88] 木庭,加藤:“動的な版の選択を行う１版先読みスケジューラ”,電子情報通信学会論文誌(D),Vol.J71-D,No.11 (1988)

[KLEI 75] L.Kleinrock:"Queueing Systems,Vol.1:Theory",Wiley(1975),（手塚,真田,中西共訳:“待ち行列システム理論(上･下)”マグロウヒル好学社）

[KOBA1 87] 小林,渡辺,真田,手塚:“確率的要素を取り扱うためのメッセージパッシングの拡張”,電子情報通信学会技術研究報告,COMP87-59 (1987-12)

[KOBA1 88a] 小林,渡辺,真田,手塚:“メッセージパッシングによる並行処理系のプロセス間関係の取扱いについて”,昭63電子情報通信学会春季全国大会,D-380 (1988)

[KOBA1 88b] 小林,渡辺,中西,手塚:“並行処理システムにおけるプロセッサ間同期方式について”,電子情報通信学会交換システム研究会,SSE88-82 (1988-7)

[KOBA1 88c] 小林,渡辺,中西,手塚:“先行制御方式におけるオーバヘッドの解析”,情報処理学会第37回(昭和63年後期)全国大会,5E-10

(1988-9)

[KOBA1 90a] 小林,小巻,中西,手塚:“タスク間の排反性に注目したタスク割当法”,電子情報通信学会コンピュータシステム研究会,CPSY90-34 (1990-7)

[KOBA1 90b] 小林,中西,手塚:“タスク多重割当法の提案”,1990年電子情報通信学会秋季全国大会,D-56 (1990-10)

[KOBA1 90c] 小林,村田,中西,手塚:“オブジェクト指向並列処理システムの提案”,1990年電子情報通信学会秋季全国大会,D-98 (1990-10)

[KOBA1 91a] 小林,渡辺,中西,手塚:“拡張メッセージ・パッシングを用いた先行制御方式のモデル化と解析”,電子情報通信学会D-I (採録決定) (1991)

[KOBA1 91b] 小林,古川,中西,手塚:“要求順サービス可能な時刻印方式について”,電子情報通信学会D-I (採録決定) (1991)

[KOBA2 86] 小林:“データベースのロック方式における並列性向上の一方式”,情報処理学会論文誌,Vol.27,No.2 (1986)

[KOHL 81] W.H.Kohler:"A Survey of Techniques for Synchronization and Recovery in Decentralized Computer Systems",ACM Computing Surveys,Vol.13,No.2 (1981)

[LI 89] 李,茨木:“時計サイトと先読みスケジューラを利用した分散データベース”,電子情報通信学会論文誌(D-I),Vol.J72-D-I,No.6 (1989)

[MASU 84] 増永:“米国における最近の分散型関係データベースシステム技術”,情報処理,Vol.25,No.5 (1984)

[MASU 87] 増永:“分散型データベースシステム”,情報処理,Vol.28,No.4 (1987)

[MITS 85] 三ツ矢,末永,奥平,安田:“マルチプロセッサによる汎用画像処理装置”,電子通信学会論文誌(D),J68-D,No.4 (1985)

[MURA1 89] 村田,小林,中西,手塚:“プログラム構成単位間通信に注目した並列性抽出に関する研究”,電子情報通信学会技術研究報告,CPSY89-58 (1989-8)

[MURA1 90] 村田,小林,中西,手塚:“並列実行性に着目したプログラム分

割と構造解析”,情報処理学会計算機アーキテクチャ研究会,ARC83-9 (1990-7)

[MURA2 89] 村岡:“並列処理技術の動向”,bit,Vol.4,No.4 (1989)

[MURA2 90] 村岡:“超並列処理コンパイラ”,コロナ社(1990)

[MURO 85] 室:“データベースの同時実行制御における直列可能性の理論”,情報処理,Vol.26,No.9 (1985)

[NAKA 82] 中川,小林,相磯:“データ駆動型離散型シミュレータKDSS-1”,電子通信学会論文誌(D),J65-D,No.3 (1982)

[NISH 90] 西尾:“データベースシステムの並行処理制御－最近の話題から－”,電子情報通信学会データベースワークショップ (1990)

[OOMO 90] 大森:“並列プログラミングの基礎”,丸善(1990)

[PADU 86] D.A.Padua and M.J.Wolf:"Advanced Compiler Optimizations for Supercomputers",Commun.ACM,Vol.29,No.12 (1986)

[POLY 87] C.D.Polychronopoules and U.Banerjee:"Processor Allocation for Horizontal and Vertical Parallelism and Related Speed up Bounds",IEEE Trans.on Computers,Vol.C-36,No.4 (1987)

[SAIT 89] 斎藤,発田:“高性能コンピュータアーキテクチャ”,丸善(1989)

[SATA 83] 佐竹,上月,西田,宮原,高島:“分散型待ち行列網シミュレータHASS-QN”,電子通信学会技術研究報告,EC83-40(1983)

[SATO 86] 佐藤,中西,真田,手塚:“並行処理型シミュレータD-SSQ”,電子通信学会論文誌,J69-D,No.3 (1986)

[SHIB 88] 柴山:“オブジェクト指向”,bit,Vol.20,No.6 (1988)

[TADA 88] 多田,近藤,宮原:“小型高並列プロセッサとその文字認識への応用”,電子情報通信学会論文誌(D),J71-D,No.8 (1988)

[TAKA 76] 高橋,五百井:“ネットワークプログラミング”,森北出版(1976)

[TAKA 89] 高橋:“並列処理機構”,丸善 (1989)

[TAKE 87] 武田,増山,茨木:“版数制限をもつ先読みスケジューラ”,電子情報通信学会論文誌(D),Vol.J70-D,No.8 (1987)

[TAKE 88] 竹内:“オブジェクト指向の指向するもの”,情報処理,Vol.29,No.4 (1988)

[TOGA 81] 戸川:“数値計算法”,コロナ社 (1981)

[TOMI 89] 富田,末吉:“並列処理マシン”,オーム社(1989)

[WATA 86] 渡辺,中西,真田,手塚:“規制先行制御方式を用いた非同期ジョブ並行処理システムの処理能力解析”,電子通信学会論文誌(D),J69-D,No.10 (1986)

[WEST 87] D.L.West and B.G.Unger:"Optimisting time warp using the semantics of abstract data types",Proc. of the Conference on AI and Simulation,(1987)

[YOKO 85] 横手,所:“並行オブジェクト指向言語Concurrent Smalltalk”,コンピュータソフトウェア,Vol.2,No.4 (1985)

[YONE 85] 米澤,松田:“Towards Object Oriented Concurrent Programming”,京都大学数理科学講究録,No.547 (1985)

[YONE 86] 米澤,柴山,J.P.Briot,本田,高田:“オブジェクト指向に基ずく並列情報処理モデルＡＢＣＭ／１とその記述言語ＡＢＣＬ／１”,コンピュウターソフトウェア,Vol.3,No.3 (1986)

[YONE 88] 米澤:“オブジェクト指向計算の現状と展望”,情報処理,Vol.29,No.4 (1988)

[YOSH 87] 吉田,所:“離散系シミュレーションの分散時刻管理”,コンピュータソフトウェア,Vol.4,No.1 (1987)

[ZHON 86] 仲,上林:“後退復帰対象を動的に決定する時刻印方式”,情報処理学会研究報告（データベース・システム）,86-DB-54-6,Vol.86,No.48 (1986)

[ZHON 89] X.Zhong and Y.Kambayashi:"Timestamp Ordering Concurrency Control Mechanisms for Transaction of Various Length",3rd International Conference Foundations of Data Organization and Argorithms(FODO) (1989)