ソフトウェア基礎論　23－5
(1987. 12. 10)

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

Unknown type message-passing for an unschedulable job

小林　真也＋　渡辺　尚＋＋　真田　英彦＋＋＋　手塚　慶一＋
Shinya kobayashi　Takashi Watanabe　Hidehiko Sanada　Yoshikazu Tezuka

＋大阪大学工学部　＋＋徳島大学工学部　＋＋＋大阪大学経済学部
Osaka University　Tokushima University　Osaka University

あらまし　本研究では、待ち行列網シミュレータや分散型データベースなどの様にプロセス間の通信が確率的な要素を含む場合には、従来のメッセージパッシングでは取り扱えないことをまず指摘する。そして、確率的な通信を扱うためにｕｎｋｎｏｗｎ型メッセージパッシングを提案し、メッセージパッシングを拡張する。さらに、ｕｎ－ｋｎｏｗｎ型を含んだ場合のプロセス同期の方法を分類する。特に矛盾の発生を許容する制御方式について述べる。

Abstract　This paper presents that it is difficult to deal with a job having probability factors, that is, queueing network simulation, by ordinarily message-passings. Then in order to deal with such a job, we propose a new type of message-passings, named unknown type. Finally we show synchronization methods for unknown type message-passings, allowing temporal contradiction.

第１章　まえがき

並行処理系における並行処理は、大規模なジョブを高速に処理していく方法として注目され、画像処理や行列計算のように処理手順が決定されているようなジョブに対してはすでに実現されている。しかし、待ち行列網シミュレーションのように乱数によって客を発生させたり、分散型データベースのようにデータベースのアクセスがいつ起こるか予め知ることができないといった、確率的な要素を持つジョブに対する実現は容易ではない。[1,2,3,4,5] そこで本稿ではこのように処理手順が予め確定していないジョブを並行処理する際のプロセス間同期の問題を明確化する。この問題を取り扱うためには並行処理系のプロセス間通信を数学的に取り扱う必要がある。そこで、プロセス間通信を数学的に取り扱うために、従来オブジェクト指向型言語におけるオブジェクト間の通信を取り扱う概念として研究されてきたメッセージパッシング[6,7]の利用を検討する。そして、従来のメッセージパッシングでは取り扱うことのできない、確率的なジョブを処理する際のプロセス間通信の取り扱いを可能とするためにメッセージ通信に非決定性を持たせたｕｎｋｎｏｗｎ型メッセージパッシングを新たに提案する。さらに、確率的要素を持つジョブを並行処理する際、処理順序に一時的な矛盾を許すプロセス間同期法[4,5]について述べる。

第２章　ジョブの分類と実行形態

並行処理は大規模なジョブを高速に実行するのに有効な手段である。並行処理系で実行されるジョブはいくつかのタスクに分割され、これらのタスクは各プロセッサに割り当てられる。各プロセスはこれらのタスクを実行していきその過程で他のプロセスとメッセージの送受信をすることによって互いに関係を持つ。また、各

プロセスはメッセージを受け取ると、そのメッセージに対応するタスクの実行を行なわなければならない。これらのタスクの実行をイベントという。しかし、イベントの順序には半順序関係があり、この順序関係を乱すことなくタスクは実行されなければならない。順序関係を保つために、各プロセス間は互いに同期を取りながら処理を進めていかなければならない。並行処理系での実行を考えると、画像処理のように各プロセス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場合には、各プロセス間の同期をとることは比較的容易である。しかし、ジョブが確率的な要素を持つ場合には、タスクの処理時刻が決まっておらず、またイベントの発生順序が処理過程に伴って決定されていき、予めイベントの発生順序は決定されていない。このようなジョブを”スケジュール不可能なジョブ”と言う。以下にジョブのスケジュール可能性と逐次処理系ならびに並行処理系でのジョブの実行形態について述べる。

２．１　スケジュール可能性

（１）　スケジュール可能なジョブ

イベント間の半順序序関係が確定しており、タスクに要する時間が決まっているいるようなジョブは予め論理時刻でいつどのようなイベントが起こるかがわかっている。そこでこのようなジョブをスケジュール可能なジョブという。

ここでは各イベントを［１］と表現し、また［１］＜［２］と表わせば［１］が［２］より以前に発生するとすることを表わすとすると、例えば

［１］＜［１１］、［２］＜［２１］、［１１］＜［２１］

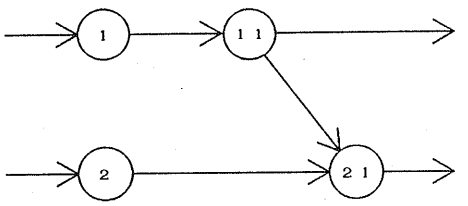


図１　イベントの順序関係
（スケジュール可能）

となり、また各タスクに対する論理時間が予め決定しているようなジョブはスケジュール可能なジョブである。このイベントの順序関係を図式で表わしたものが図１である。

（２）　スケジュール不可能なジョブ

あるイベントの次に起こるイベントが確定していなかったり、そのイベントにかかる論理時刻が、他のイベントによって決定されるようなジョブをスケジュール不可能なジョブという。

具体的には図２のように、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るような場合である。この場合、イベント［１］が終了するまで［１１］と［１２］の何れが発生するか決定していない。このようなジョブの実行は、実行開始以前にその実行過程においてどのようなイベントが発生するかは不明である。

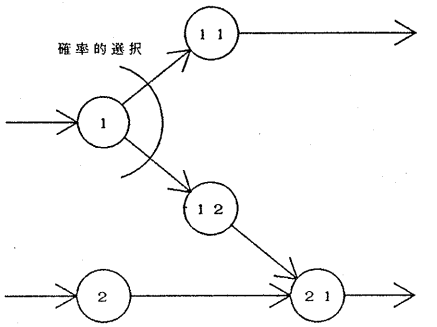


図２　イベントの順序関係
（スケジュール不可能）

２．２　逐次処理系での実行

（１）　スケジュール可能なジョブに対する実行

図１に示したようなイベントの発生順序を持つジョブをシングルプロセッサによって逐次処理系で実行したときに、プロセッサにおけるイベントの発生順を示したものが図３である。図３に示す３通りのイベント発生順であればどれであっても［１］＜［１１］、［２］＜［２１］、［１１］＜［２１］を満たすことができるので正しい結果を得ることができ、またジョブの実行以前にこの発生順を決めておくことができる。

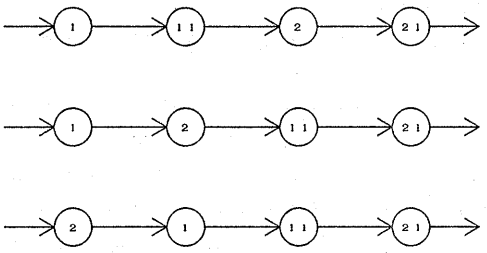


図３　プロセッサにおけるイベント発生系列
（スケジュール可能）

（２）スケジュール不可能なジョブに対する実行

図２に示したようなイベントの発生順序を持つスケジュール不可能なジョブを逐次処理すると、イベント［１１］が発生する場合とイベント［１２］が発生する場合の２通りに分けることができる。図２に示すジョブを実行するには［２２］のイベントが始まる以前に［２］のイベントが終了していなければならない。また、もし［１２］のイベントが発生するならば、［１２］のイベントは［２１］のイベントの開始以前に終了していなければならない。したがって、イベント［２１］は、［１２］が起こるか起こらないかにかかわらず、［１２］が起こるか起こらないかが明白となる、つまり［１］のイベントが終わるまでは、始まることはできない。また、もし［１２］が起こるならば、それが終了するまでは待たなければならない。したがって、各イベントの発生順序関係は以下に示すように条件文を用いて表わすことができる。

```
［２］＜［２２］
ｉｆ　［１１］が起こる
　　［１］＜［１１］
ｅｎｄｉｆ
ｉｆ　［１２］が起こる
　　［１］＜［１２］
　　［１２］＜［２２］
ｅｎｄｉｆ
```

状態	開始可能なイベント	次に開始するイベント
1	[1]、[2]	[1]
2	[2]、[11]	[2]
3	[11]、[21]	[21]
4	[11]	[11]
:	:	:

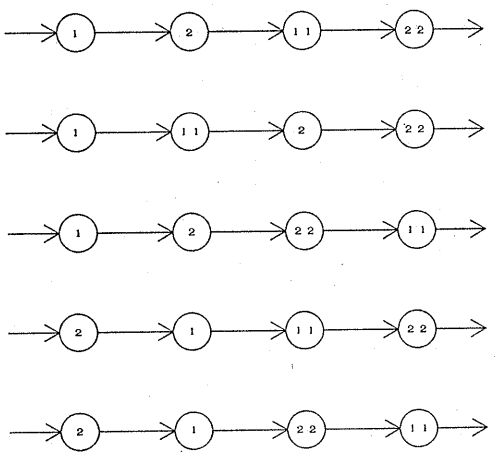
(a)　[11]が発生する場合

状態	開始可能なイベント	次に開始するイベント
1	[1]、[2]	[1]
2	[2]、[12]	[2]
3	[12]	[12]
4	[22]	[22]
:	:	:

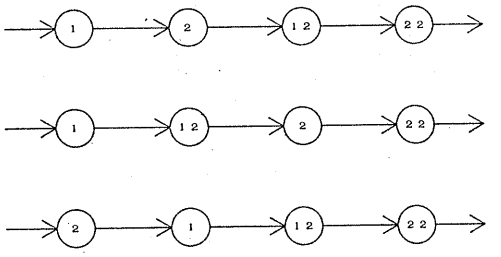
(b)　[12]が発生する場合

図4　イベント処理手順

そこで、プロセッサがどのようにしてこの条件を満たしながら実行を進めていくかを考えてみると図４のようになる。図４はあるイベントが終了した時点における次に開始可能なイベントを示す欄と次に開始するイベントを示す欄の２つからなる。もちろん次に開始するイベントは開始可能なイベントを示す欄にも含まれ、プロセッサが開始可能なイベントの中から任意に選びだしたイベントである。図４の（ａ）は［１１］が選ばれた場合の例で、（ｂ）は［１２］が選ばれた場合の例である。図４の（ｂ）では状態３に注意しなければならない、この時イベント［２］は終了しているが［１２］がまだ終了していないので［２１］はまだ開始可能となっていない。このように、このイベント発生順序関係の条件を満たすようなプロセッサにおけるイベント発生順序を全て考えると、［１１］が発生する場合と［１２］が発生する場合の２種類に分類でき、それぞれの場合を図５に示す。



（ａ）　［１１］が発生した場合



（ｂ）　［１２］が発生した場合

図５　プロセッサにおけるイベント発生系列
（スケジュール不可能）

図５に示す順序関係であれば必ず正しい結果を得ることができ、また、このイベント順序関係は予め決定していないが、プロセッサが処理を行なうにしたがっておのずと決定されていく。

２．３　並行処理系での実行

ここでは以下に示すような仮定を持つ並行処理系を考える。

（１）　ジョブはいくつかのタスクに分割される。

（２）　タスクは各プロセッサに割り当てられる。

（３）　各タスクは実行されるべき論理時刻がある。

（４）　各プロセスは論理時刻の増加順にタスクの処理を行なう。

（５）　各プロセス共通の論理時刻を示す時計はない。

（６）　各プロセスはそれぞれ独立した論理時刻を持つ。

（７）　各プロセスは送信時の論理時刻を付加されたメッセージを他のプロセッサに送信する。

（８）　メッセージを受け取ると、各プロセスはメッセージによって要求されたタスクを実行する。

このような並行処理系でスケジュール可能なジョブと不可能なジョブの実行について考えてみる。

（１）　スケジュール可能なジョブに対する実行

図１に示すような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行処理系で実行する。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に割り当てられたとすると、図６に示すように［１］、［１１］はプロセッサ１のイベント発生系列に含まれ、［２］、［２１］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におけるイベントの発生についてみてみると、まずイベント［２］が発生し終了するがこの次のイベントである［２１］はプロセッサ１のイベント［１１］が終了するまでは開始できないことが予めわかっている。そこでこのようなジョブを並行処理する場合には［１１］と［１２］の発生順序に矛盾が生じないように、プロセッサ１と２の間に処理の開始時刻の同期をとること

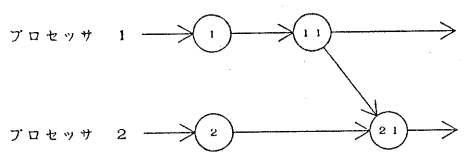


図６　プロセッサにおけるイベント発生系列
（スケジュール可能）

になる。この同期は、プロセッサ１が［１１］が終了すると何らかの手段（メッセージの送信等）でプロセッサ２に対して［１１］の終了を伝えることで行なえる。プロセッサ２はイベント［２］が終了した時点でプロセッサ１から［１１］の終了が伝えられていればただちに、［２１］を開始するが、まだ伝えられていなければ、プロセッサ１から伝えられるのを待って、［２１］を開始する。図７はメッセージによってプロセッサ１から２に対し、［１１］の終了

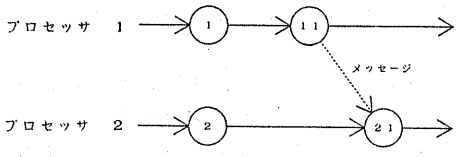


図７　プロセッサ間の同期
（スケジュール可能）

が伝えられプロセッサ２が［２１］を開始する様子を表わしている。

（２）スケジュール不可能なジョブに対する実行

次に図２のようなイベント発生順となるジョブを２台のプロセッサからなる並行処理系での実行を考える。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］に対するタスクがプロセッサ２に割り当てられたとすると。イベント［１］および、［１１］又は［１２］の何れか一方の２つのイベントはプロセッサ１のイベント発生系列に、［２］と［２１］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みてみると、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］をただちに開始することはできない。なぜなら、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ためである。したがってプロセッサ２はプロセッサ１から［１２］の終了を伝えられるまでは［２１］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］のほう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］の終了を伝えられることがなく、そこで動作を停止してしまうことになる。このようにスケジュール不可能なジョブを並行処理するには、スケジュール可能な場合には無い問題が生じる。

第３章　メッセージパッシング

第２章で述べた並行処理系におけるプロセッサ間の通信はメッセージパッシング[6,7]の概念を用いると数学的に取り扱うことができる。メッセージパッシングの考え方は、従来オブジェクト指向型言語などの分野で使われてきたものであり、メッセージを送る送信プロセスと、受け取る受信プロセスの関係を取り扱うものである。メッセージパッシングを並行処理系のプロセッサ間の通信の取り扱いに用いるためには、既存のタイプだけでは完全に取り扱うことはできない。以下では、まず既存のタイプのメッセージパッシングについて述べ、さらに新しいタイプのメッセージパッシングを提案する。

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。

プロセスＡがプロセスＢからのメッセージを受け取った後、

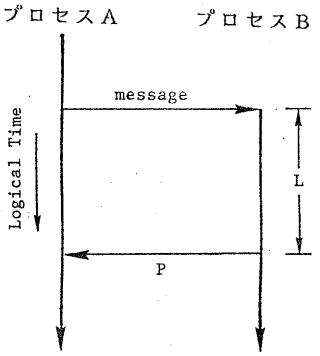
（１）　プロセスＡがプロセスＢに対してメッセージを返すかどうか。

（２）　メッセージを返す場合に、プロセスＡがメッセージを受け取ってから返すまでの論理時間間隔。

論理時刻はイベントが持っている仮想の時刻であり、イベントの発生はこの論理時刻によって順序付けられる。また、ある論理時刻と他の論理時刻の間隔を論理時間という。

表１はメッセージパッシングを、メッセージが返される確率とその論理時間間隔で分類したものである。表１において”ｋｎｏｗｎ”は、プロセスＢがプロセスＡからメッセージが送られる以前に送られる論理時刻がわかっていることを意味する。

表１における６種のメッセージパッシングのうちｐａｓｔ型、ｎｏｗ型、ｆｕｔｕｒｅ型はすでに提案されているメッセージパッシングであり、ｕｎｋｎｏｗｎ型メッセージパッシングが今回提案する新しいタイプのメッセージパッシングである。以下にこれら６種のメッセージパッシングについて説明する。



p	L	type
0	---	past
1	known	now,future
	unknown	unknown-I
(0,1)	known	unknown-II
	unknown	unknown-III

表1　メッセージパッシングの分類

（1）　past型メッセージパッシング

P＝0（Pはプロセスプロセスからプロセスにメッセージが返される確率を意味する）

プロセスBからプロセスAにメッセージが送られた後、プロセスAからプロセスBに対しメッセージが返されることはないので、プロセスBはプロセス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。このタイプのメッセージパッシングをpast型という。

（2）　now型およびfuture型メッセージパッシング

P＝1、Lが決定している。（Lはプロセスがメッセージを受け取ってから送り返すまでの間の論理時間間隔）

このタイプはプロセスAがメッセージを必ず返し、また（論理時刻で）いつ返すかもわかっている。now型とpast型は以下のような違いがある。

now型はL＝0である、つまりプロセスBはメッセージが送り返されるまで論理時刻を更新しない。now型メッセージパッシングは関数呼び出しや手続き呼び出しに類似しているが、プロセスAがメッセージを返した後停止しない点で異なる。

一方　future型メッセージパッシングではL＞0となっている。つまりプロセスBは、プロセスAからメッセージが返されるまでの間になんらかの処理を行ない論理時刻の更新を行なう。

（3）　unknown型メッセージパッシングI

P＝1、Lが決定されていない。プロセスAからメッセージが返されることは確定しているが、プロセスBはメッセージがいつ返されてくるのかわからない。

（4）　unknown型メッセージパッシングII

P＝（0，1）、Lは決定されている。プロセスAからメッセージが返されるかどうかはわからないが、プロセスBはメッセージがもし返されるならば、返されるであろう時刻を知っている。

（5）　unknown型メッセージパッシングIII

P＝（0，1）、Lが決定されていない。プロセスAからメッセージが返されるかどうかわからないし、もし返されるとしても、返されるであろう時刻もわからない。

スケジュール可能なジョブはどのようなイベントがいつ起こるかが予めわかっているため、メッセージが送られるかどうか、またいつ送られるかどうかが予め決まっているnow型、past型、future型メッセージパッシングで取り扱うことができる。しかし、スケジュール不可能なジョブはどのイベントが起こるかわからないため、並行処理系で、他のプロセッサからメッセージが送られてくるかどうかわからないし、またいつ送られるかもわからないため、これら従来の3つのタイプのメッセージパッシングでは取り扱う事ができず、確立的な要素を持ったunknown型メッセージパッシングを用いなければならない。

第4章　unknown型メッセージパッシングに対する同期法

並行処理系においては各プロセッサは他のプロセッサと独立に動作する、しかしとくにスケジュール不可能なジョブに対しては、イベントの発生順序関係の条件を満たすために、プロセ

ッサ間で何らかの同期とる必要がある。ここでは、スケジュール不可能なジョブに対応するｕｎｋｎｏｗｎ型のメッセージパッシングの同期法について、イベントの発生順序における矛盾をまったく引き起こさない方法と、一時的な矛盾を許容する方式について述べる。

４．１　矛盾が起こらない方法

（１）　ｕｎｋｎｏｗｎ型Ｉに対する同期法

このタイプのメッセージパッシングはメッセージが送られてくることは確定しているがいつ送られてくるかはわからない。このタイプのメッセージパッシングに対しては以下のような同期法が可能である。

ａ）同期法Ｉ

プロセスＡからメッセージが返されるまでプロセスＢが論理時刻を更新するのを禁止する。例えば図８では、プロセスＡが実時刻Ｔにメッセージをプロセスに送り返すまでプロセスＢは論理時刻を更新することを許されない。この同期法ではプロセス間の同期を完全にとることが可能であるが、プロセスＢは実時刻０からＴの間何もしない（アイドル状態）。

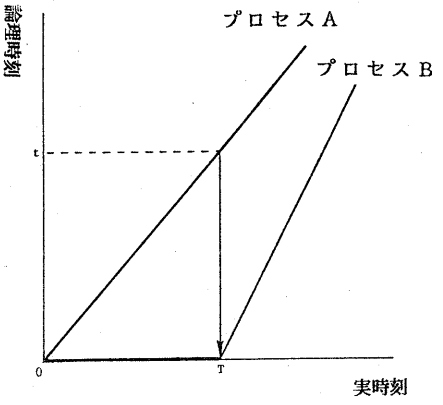


図８　同期法Ｉ（ｕｎｋｎｏｗｎ型Ｉ）

ｂ）同期法Ⅱ

プロセスＡは適当な時刻にプロセスＡの論理時刻を持った制御メッセージをプロセスＢに対し送る、プロセスＢは制御メッセージを受け取ると制御メッセージの示す論理時刻まで進むことが許される。図９では実時刻Ｔ１、Ｔ２、Ｔ３に論理時刻ｔ１、ｔ２、ｔ３の論理時刻を持った制御メッセージがプロセスＡからプロセスＢに対し送られる。例えばプロセスＢはプロセスＡから実時刻Ｔ１に論理時刻ｔ１の制御メッセージを受け取ると、論理時刻ｔ１まで進むことが許可される。この同期法は同期法Ｉに対しアイドル時間を少なくできる。

これら２つの同期法では矛盾は発生しない。

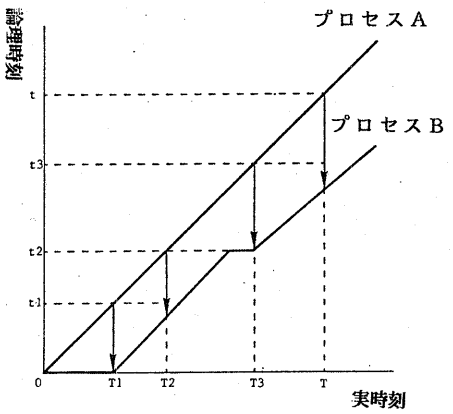


図９　同期法Ⅱ（ｕｎｋｎｏｗｎ型Ⅱ）

（２）　ｕｎｋｎｏｗｎ型Ⅱに対する同期法

このタイプのメッセージパッシングはメッセージが送られるかどうかはわからないが、もし送られるならその時刻はわかっている。

このタイプのメッセージパッシングに対する同期法はｎｏｗ型やｆｕｔｕｒｅ型に対する同期法と同じである。プロセスＢはメッセージが送られてくるであろう論理時刻まで進むことが許される、プロセスＡはメッセージを送るであろう論理時刻に達すれば真のメッセージか制御メッセージをプロセスＢに送る、プロセスＢはプロセスＡから真のあるいは制御メッセージをを受け取ると先に進むことが許される。

（３）　ｕｎｋｎｏｗｎ型Ⅲに対する同期法

このタイプのメッセージパッシングはプロセスＡからプロセスＢにメッセージが返されるかどうかわからないし、またもし返されるとしてもいつ返されるかわからない。このタイプのメッセージパッシングに対する同期法の１つとしては、ｕｎｋｎｏｗｎ型Ｉに対する同期法Ｉの様に制御メッセージを送るものがある。またこれ以外にもタイムドリブン方式や先行制御方式がある。

ａ）時刻駆動方式

時刻駆動方式では並行処理系はすべてのプロセスに共通なグローバルクロックを持っている。グローバルクロックは一定時間間隔ごとにすべてのプロセスに信号を送り、各プロセスは信号を受け取るとその時間間隔に処理されるタスクを同時に実行する。この同期法は同一時間間隔内のタスクの実行順序は考慮しないため、計算結果に誤差を含み、時間間隔を長くとればとるほど同時に動作しているプロセス数は増えるが、同一時間間隔に含まれるタスクは同一論理時刻であるとみなされるため誤差が大きくなる。一方、この時間間隔を短くすれば誤差は減るが、システムの実行速度は低下する。

４．２　一時的な矛盾を許す方式

４．１で述べた同期法はどれも、イベントの発生順序にまったく矛盾を生じる事が無い、しかし一時的な矛盾の発生を許しそれを解消しながら処理を進めていく方法もある。以下にこの方法の１つである先行制御方式について述べる。

４．２．１　先行制御方式

先行制御方式[4.5]では、各プロセスは他のプロセスとまったく独立に処理を行なっていく、そのため他のプロセスからメッセージが送られてきたときに、そのメッセージの受信以前には行なってはならないタスクの実行をしており、イベントの発生順序に矛盾が生じる場合がある。その場合には矛盾となったプロセスはプロセスの状態をメッセージ受信前の状態まで戻し矛盾を解消する、これをキャンセル処理という。プロセスは矛盾を解消すると、その状態から再び処理を再開する。

第２章で考えたように、図２に示すようなイベントの発生順序関係を持つスケジュール不可能なジョブを並行処理系で実行した際に、同期法として先行制御方式を用いた場合に、各プロセッサの実時刻と論理時刻の関係を示したのが図１０である。図１０では、プロセッサ２はイベント［２］が終了すると、ただちにイベント［２１］を開始している、しかし実時刻Ｔにプロセッサ１からイベント［１２］の終了を示すメッセージが送られてきたので、プロセッサ２はその状態をイベント［１２］開始以前まで戻し、この状態から再処理を行っている。

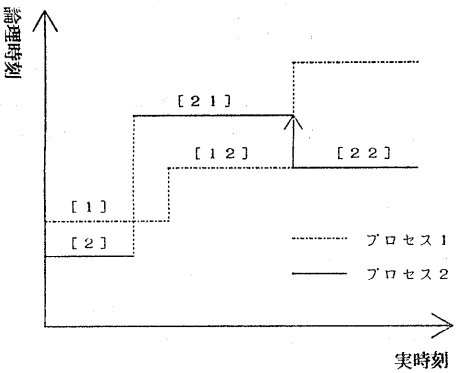


図１０　先行制御方式による同期

先行制御方式ではこのようにしてプロセスは矛盾が生じると、その状態を矛盾発生以前に戻し矛盾を解消していくことにより、ジョブの処理全体としての矛盾を含まないようにしている。同様の制御方法としては、Ｔｉｍｅ　Ｗａｒｐ　Ｍｅｃｈａｎｉｓｍがある。

４．２．２　先行制御方式の高速化

先行制御方式では矛盾を生じなければ、各プロセスは他のプロセスとは関係なしに処理を進めていくため、ｎ個のプロセスがあればｎ倍の速度を得ることができる。しかし、矛盾が生じるとキャンセル処理を行なわなければならないため、高速化の妨げとなる。したがって、先行制御方式ではいかにキャンセル処理を少なく押さえるかが高速化の鍵となる。キャンセル処理を少なく押さえるには、矛盾の発生自体を少なくする方法と、矛盾が生じた際にキャンセルする量を少なくする方法があるが、これらは完全に分離されるものではなく、どちらに主眼を置くかという問題で、通常は一方を改善すると他法も改善される。以下にこれらの改善法を示す。

a）規制先行制御方式

各プロセスに対し、並行処理系全体の中で最も遅れているプロセスよりも先行できる量に規制を設け、それ以上は進めないようにする方法である。この方法では、各プロセスの論理時間を一定枠の中におさめることにより、矛盾の発生を押さえることを目的としており、この方法での改善成果はすでに確認されている。[4、5]

b）最適分配法

これはキャンセル処理量を少なくすることに主眼を置いた改善法で、キャンセル処理量を少なくするようにタスクの割り当てを行なう方法であり、具体的にはプロセスの進み具合の均一化とメッセージ送受信量の最小化を行なうタスク分配法である。

・プロセスの進み具合の均一化

各プロセスの進み具合を均一化することにより、プロセス間の進み具合の差を小さくし、矛盾が生じた際のキャンセル量を減らしている。また、矛盾の発生を押さえることも期待できる。

・送受信の最小化

メッセージの送受信がなければ矛盾は生じることはない。そこでメッセージの送受信を少なくするようにタスクを割り当てることで、矛盾の発生を押さえる。

第５章　まとめ

本稿では、まず確率的な要素を持つスケジュール不可能なジョブを並行処理する際の問題を示した。次に、スケジュール不可能なジョブを並行処理系で処理する際のプロセッサ間の関係を取り扱えるように、従来からあるメッセージパッシングに対しｕｎｋｎｏｗｎ型メッセージパッシングを導入した。また、このｕｎｋｎｏｗｎ型メッセージパッシングに対するいくつかの同期法を示した。今後の課題としてはスケジュール不可能なジョブを効率よく実行するための種々のプロセッサ同期法の効率評価を行なう必要がある。

［参考文献］

[1] 稲守，戸田：”複数マスタプロセッサを用いた並列型通信網トラヒックシミュレータの評価，”信学論（Ｂ），J65-B,1,pp.22-29(1985)

[2] 佐竹，上月，西田，宮原，高島：”分散型待行列網シミュレータ，”信学技報,EC83-40(1983)

[3] 中川，小林，相磯：”データ駆動型離散型シミュレータKDSS-1，”信学論（Ｄ），J65-D,3,pp.386-393(1982)

[4] 佐藤，中西，真田，手塚：”並列処理待ち行列網シミュレータD-SSQ,”信学論（Ｄ）Vol.J69-D,No.3(1986)

[5] 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，”信学論（Ｄ）Vol.J69-D,No.10(1986)

[6] Hewitt,C. :”Viewing Control Structures as Patterns of Passing Messages,”Artificial Intelligence,Vol.8,pp.323-364(1977)

[7] 米沢，松田：”Towards object Oriented Concurrent Programming,”京都大学数理科学講究録547,pp.1-2(1985)