数理解析研究所講究録
第625巻 1987年 196-205

ＦＩＦＯキューを同期手段とする並列プログラムの実行における同期のためのオーバーヘッドの削減について

九工大情報工学科　永松　正博　（ Masahiro Nagamatsu ）
九工大情報工学科　有田　五次郎（ Itsujiro Arita　　）

１．まえがき

MIMD型の並列処理システムでは、セマフォー等を使用した同期手段により、先行制約が与えられたタスク集合（並列プログラム）の実行を行う。並列実行される各タスクの実行時間が同期操作のために要する時間に比べて大きい場合には、同期のためのオーバーヘッドはあまり問題とならず、十分効率の良い並列処理が行える。しかし、各タスクの実行時間が小さく、タスクの個数も多い高多重並列処理においては、この方法では、同期のためのオーバーヘッドが大きくなり、並列処理の効果が上がらなくなる。従って、同期操作を高速にすることや、同期操作の回数を減らすことが必要となる。

有田[(1),(2),(3)]らは、各プロセッサがFIFOキューを持ち、各タスクはこれらのFIFOキューに投入され、各プロセッサは自分のFIFOキューから先着順にタスクの実行を行っていくような実行機構（FCFS）において、ある種の先行関係が、同期操作を行わなくても、並列プログラムの実行にともない誘導されることを示している。

本論文は、FCFSによる実行にともない誘導される先行関係のグラフ理論的な特性付けを行い、同期操作の回数ができるだけ小さい並列プログラムを得るためのアルゴリズムを示す。

２．定義

Aを集合、BをA上の半順序関係とする。$(t_1,t_2)\in B$であることを$t_1\leq_B t_2$と書き、$t_1\leq_B t_2$かつ$t_1\neq t_2$であることを$t_1<_B t_2$と書く。$t_1<_B t_2$であり、かつ、$t_1<_B t<_B t_2$であるtが存在しないとき、t_1はt_2を被覆するという。A上の２項関係 $\{(t_1,t_2)\mid t_1$ は t_2 を被覆する$\}$ をBの被覆関係という。

タスクの有限集合をT、タスク間の先行制約を表す半順序関係をRとする。各タスクtに対して$t_s\leq_R t$であるタスクt_sが存在すると仮定する。t_sを開始タスクと呼ぶ。Tを頂点の集合とし、T×Tの部分集合を枝の集合とし、有向閉路を持たず、t_sの入次数が0であり、かつ、t_s以外の各タスクの入次数が1以上であるような有向グラフを並列プログラムという。また、t_s以外の各タ

スクの入次数が 1 であるような並列プログラムを待ちなし並列プログラムという。

$E \subseteq T \times T$ であるとき、E の要素を E枝と呼ぶ。$(t_1,t_2) \in E$ のとき t_1 を t_2 の E親、t_2 を t_1 の E子、という。$(t_1,t_2),(t_2,t_3),\cdots,(t_{k-1},t_k) \in E$ $(k \geq 1)$ である道 $p=(t_1,t_2,\cdots,t_k)$ を t_1 より t_k に到る E道という。道に含まれる枝の個数をその道の長さといい、length(p) と書く。t_1 から t_2 に到る E道が存在するとき、t_1 を t_2 の E先祖、t_2 を t_1 の E子孫という。更に、$t_1 \neq t_2$ であるとき、t_1 を t_2 の真の E先祖、t_2 を t_1 の真の E子孫という。

タスク t に対して、t の E深さ depth(t:E)、E高さ height(t:E) を次のように定義する。ただし、P_1 は t へ到る E道の集合、P_2 は t を出発点とする E道の集合である。

$$\mathrm{depth}(t:E)=\max_{p \in P_1} \mathrm{length}(p)$$

$$\mathrm{height}(t:E)=\max_{p \in P_2} \mathrm{length}(p)$$

各タスク t に対して、t に入る E枝の本数を t の E入次数といい、$d^-(t:E)$ で表す。また、t から出る E枝の本数を t の E出次数といい、$d^+(t:E)$ で表す。$d^-(t:E) \geq 2$ であるタスクを、E合流タスクという。

3．並列プログラムの実行

本章では、有田[3]に提案されているような FIFOキューを持つ並列処理システムで、並列プログラム G=(T,E) を実行することを考える。

プロセッサは全て同一であるとする。各プロセッサは FIFOキューを持っており、そのキューに到着した順にタスクの処理を行う(FCFS, First Come First Served)。各タスクの実行プロセッサの割り当ては、実行前になんらかの方法で決定されているとする。タスク t が割当てられているプロセッサを pr(t) で表す。pr をプロセッサ割当関数と呼ぶ。なお、開始タスクは特別なタスクであり、t_s 以外の各 t に対して、$pr(t_s) \neq pr(t)$ であると仮定する。この仮定は後の議論を行いやすくするためだけのものであり、本質的なものではない。

また、枝順序関数と呼ばれる関数 $f:(T \times T) \rightarrow \{1,2,\cdots,|T|\}$ が与えられているとする。f は 各タスク t に対して、t から出る枝の間に順序を付けるためのものであり、$\{f((t,t_1))\mid t_1 \in T\}=\{1,2,\cdots,|T|\}$ であるとする。$f((t,t_1))$ を単に $f(t,t_1)$ と書くことにする。

以下に述べるアルゴリズムは、与えられた並列プログラム G=(T,E)、プロセッサ割当関数 pr、枝順序関数 f に対して、その実行を行うものである。G の各 E合流タスク t は、セマフォ変数 V_t をもち、実行前 $V_t=d^-(t:E)$ で初期化されているとする。

[アルゴリズム　FCFS(G,pr,f)]

```
procedure FCFS(G,pr,f);
  begin
  ts をプロセッサpr(ts) の FIFOキューに投入する;
  各プロセッサは、並列に以下を実行する
    begin
    実行中のタスクがなければ、自分のFIFOキューの先頭からタスクを1個取
    り出し（そのタスクを t とする。FIFOキューが空のときは、FIFOキューに
    タスクが投入されるのを待ってから取り出す）、その実行を行う;
    t の実行が終了すると、tの各 E子u に対して、f(t,u) の 小さいものから
    順に以下を実行する
      begin
      if (u が E合流タスク) then
        begin
        Vu:=Vu-1;
        if (Vu=0) then  u をプロセッサpr(u) の FIFOキューに投入する;
        end
      else  u をプロセッサpr(u) の FIFOキューに投入する。
      end;
    end;
  end;                                                          ■
```

各タスクt に対して、実行を開始する時刻を実行開始時刻、そのタスク本来の仕事が終了する時刻を実行終了時刻、そのタスクのすべての E子に対する同期操作を終了する時刻を処理終了時刻と呼ぶ。E子に対する同期操作とは、その E子が E合流タスクであるときは、セマフォー操作と FIFOキューへの投入であり、E合流タスクでないときは、FIFOキューへの投入のみである。

各タスクt の実行時間、及び、同期操作の時間を任意に定めて、実際に アルゴリズムFCFS(G,pr,f) を実行した結果を、ひとつの実行例と呼ぶ。図1に実行例を示す。(a) が並列プログラムであり、(b) がそのタイミングダイヤグラムである。(a) において各タスクt の横に付けられた( )の中の数字は t を実行するプロセッサpr(t) を示す。また、各枝に付けられた数字は枝順序関数f の値を示す。(b) において、中にタスク番号が書かれた四角はそのタスクの実行を表し、中に矢印が書かれた四角は FIFO キューへの投入操作を表し、また、ハッチングされた四角はセマフォー操作を表す。

このアルゴリズムは、通常のセマフォーによる同期の実現方法のひとつである。一般に並列処理においては、各タスクの実行時間が同期操作に要する時間に比べて大きく、かつ、タスクの個数も少ない場合は、十分に並列化の効果が期待できる。しかし、タスクの個数が多く、かつ、各タスクの実行時間が短い場合は、同

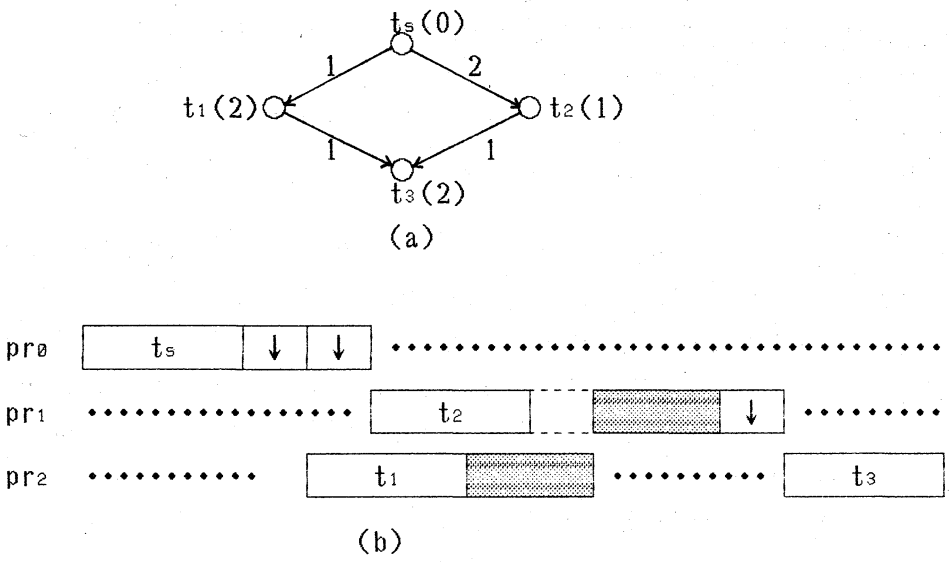

図１　FCFS による実行例

期操作のオーバーヘッドが大きくなり、並列化の効果がなくなってしまう。また、同じセマフォー変数に対する操作や、同じ FIFOキューへのタスクの投入は排他的にしか実行できず、このため同期操作に要する時間を大きくしてしまう。本論文の目的は、同期操作の回数（すなわち、E枝の個数）を少なくすることにより、同期のためのオーバーヘッドを減少させることである。

４．ＦＣＦＳにより誘導される先行関係

G=(T,E) を並列プログラム、pr をプロセッサ割当関数、f を枝順序関数とする。次のように定義される T 上の２項関係 $R_{ipr}(G,pr,f)$ を G,pr,f に対して FCFS により誘導される先行関係(IPR, Induced Precedence Relation)という。

$R_{ipr}(G,pr,f)$= {(t_1,t_2) | 任意の実行例において、t_1 の実行終了時刻よりも t_2 の実行開始時刻の方が後である。} ∪ {(t,t) | t∈T}

G、pr、f が明らかな場合は、$R_{ipr}(G,pr,f)$ を単に R_{ipr} と書くことにする。

次の２つの補題は明かである。

［補題１］ $R_{ipr}((T,E),pr,f)$ は T 上の半順序関係である。 ■

［補題２］ $E^* \subseteq R_{ipr}((T,E),pr,f)$ である。 ■

タスクの集合 T と、その先行制約を表す半順序関係 R が与えられたとき、E を R の被覆関係（すなわち、$E^*=R$ であるような最小な E ）として、並列プログラム G=(T,E) を作り、適当なスケジューリングアルゴリズムを使用して、プロセッサ割当関数 pr を決定し、また、枝順序関数 f を適当に決定して、FCFS

により実行を行えば、R に違反することなく、T の実行を行うことができる。これが従来行われている並列プログラムの実行方法である。有田らは FCFS による実行において、同期操作のうちのいくつかが不用であること、すなわち、$E^* \subsetneq R$ であるような並列プログラム G=(T,E) でも、R を満足するような実行が可能であることを示している。本論文は、この考え方を拡張し、かつ、論理的な基盤を与えることを目的とする。

与えられた先行制約 R を満足するためには、$R \subseteq R_{ipr}$ であればよい。同期操作の回数、すなわち、並列プログラムの枝の個数を減少させるには、次に示すアルゴリズムにより、不用な枝、すなわち、$R \subseteq R_{ipr}((T,E-\{e\}),pr,f)$ であるような枝 e を探し出し、それを除くことを繰り返せばよい。ただし、E の初期値としては、$R \subseteq R_{ipr}((T,E),pr,f)$ であるようなものでなければならない。

[アルゴリズム FindMinPP_0(G=(T,E),pr,f)]

```
procedure FindMinPP_0(G=(T,E),pr,f);
  begin
  while (R⊆Ripr((T,E−{e}),pr,f) であるような e∈E が存在する) do
    E:=E−{e};
  end;                                                    ■
```

このアルゴリズムは、与えられた G=(T,E),pr,f に対して、E から不用な枝を順次除き、R を満足するために必要な極小の枝集合にする。しかしながら、このアルゴリズムを実行するには、R_{ipr} を計算する方法が必要になる。結論から先に述べると、本論文では、R_{ipr} を計算する方法を与えてはいない。R_{ipr} の代用となる集合を定義し、それを計算することにより不用な枝を除くアルゴリズムを示す。以下、このための準備を行う。

T×T 上の２項関係 $B_{ipr}(G,pr,f)$、および、T 上の２項関係 $D_{ipr}(G,pr,f)$ を次のように定義する。

$B_{ipr}(G,pr,f)$= {$((t_1,t_3),(t_2,t_4))$ | $(t_1,t_3),(t_2,t_4) \in E$ であり、
任意の実行例において、t_1 が t_3 に対する同期操作を終了する時刻よりも、 t_2 が t_4 に対する同期操作を開始する時刻の方が後である。}

$D_{ipr}(G,pr,f)$= {(t_1,t_2) | 任意の実行例において、t_1の処理終了時刻よりも t_2の実行開始時刻の方が後である。} ∪ {(t,t) | t∈T}

つぎの２つの補題は明かである。

[補題３] $D_{ipr}((T,E),pr,f)$ は T 上の半順序関係である。 ■

[補題４] $D_{ipr}(G,pr,f) \subseteq R_{ipr}(G,pr,f)$ である。 ■

[A_{ipr}、E_{ipr} の定義] G=(T,E) とする。T×T 上の２項関係 $A_{ipr}(G,pr,f)$、および、T 上の２項関係 $E_{ipr}(G,pr,f)$ を次のように定義する。

① $(t_1,t_2),(t_2,t_3) \in E$ ならば、

$((t_1,t_2),(t_2,t_3))\in A_{ipr}(G,pr,f)$ である。

② $(t_1,t_2),(t_1,t_3)\in E$、かつ、$f(t_1,t_2)<f(t_1,t_3)$ ならば、
$((t_1,t_2),(t_1,t_3))\in A_{ipr}(G,pr,f)$ である。

③ $(t_1,t_3),(t_2,t_4)\in E$、かつ、$(t_1,t_2)\in E_{ipr}(G,pr,f)$ ならば、
$((t_1,t_3),(t_2,t_4))\in A_{ipr}(G,pr,f)$ である。

④ $t_1 \neq t_2$、$t_1 \neq t_5$、かつ、
t_1 の各 E親 u_1 に対して、
1) $t \neq t_1$、
2) $(u,t)\in E$、
3) (u,t) は (u_1,t_1) の $A_{ipr}(G,pr,f)$子孫、
4) $pr(t)=pr(t_1)$、かつ、
5) t は t_2 の E先祖
であるような u,t が存在するならば、
$(t_1,t_2)\in E_{ipr}(G,pr,f)$ である。

⑤ 上記の①～④で $A_{ipr}(G,pr,f)$、$E_{ipr}(G,pr,f)$ の
要素となるもの以外は $A_{ipr}(G,pr,f)$、$E_{ipr}(G,pr,f)$
の要素でない。 ■

G,pr,f が明らかな場合、$A_{ipr}(G,pr,f)$、$B_{ipr}(G,pr,f)$、$D_{ipr}(G,pr,f)$、$E_{ipr}(G,pr,f)$ を、それぞれ、単に A_{ipr}、B_{ipr}、D_{ipr}、E_{ipr} と書くことにする。

［補題５］ $((t_1,t_3),(t_2,t_4))\in A_{ipr}$ ならば、$((t_1,t_3),(t_2,t_4))\in B_{ipr}$ である。また、$(t_1,t_2)\in E_{ipr}$ ならば、$(t_1,t_2)\in D_{ipr}$ である。

（証明） A_{ipr}、E_{ipr} の定義において、①～④の各条件部において補題が成立すれば、結論部において新たに A_{ipr}、E_{ipr} の要素となるものに対しても補題が成立することを示すことにより、帰納的に証明する。

①の場合、$(t_1,t_2),(t_2,t_3)\in E$ であるので、t_2 が t_3 に対する同期操作を開始する時刻より前に、t_1 が t_2 に対する同期操作を終了していなければならない。故に、明らかに、$((t_1,t_2),(t_2,t_3))\in B_{ipr}$ である。

②の場合、$(t_1,t_2),(t_1,t_3)\in E$、かつ、$f(t_1,t_2)<f(t_1,t_3)$ であるので、t_1 が t_2 に対する同期操作を終了した後に、t_3 に対する同期操作を行う。故に、明らかに、$((t_1,t_2),(t_1,t_3))\in B_{ipr}$ である。

③の場合、帰納法の仮定より、$(t_1,t_2)\in D_{ipr}$ であるので、t_1 の処理が終了した後に、t_2 の実行が開始される。従って、t_2 が t_4 に対する同期操作を開始する時刻より前に、t_1 が t_3 に対する同期操作を終了している。故に、$((t_1,t_3),(t_2,t_4))\in B_{ipr}$ である。

最後に、④の場合を考える。任意の実行例をαとする。このαにおいて、t_1 はその E親 u_1 により、$pr(t_1)$ の FIFO キューに投入されるとする。この u_1 に対しても、A_{ipr}、E_{ipr} の定義の④で述べるような u,t が存在する。帰納法の

仮定より、(u,t) は (u_1,t_1) の B_{ipr}子孫、すなわち、u_1 が t_1 を $pr(t_1)$ の FIFO キューに投入する時刻よりも、t が pr(t) の FIFO キューに投入される時刻の方が後である。 ところが、 $pr(t)=pr(t_1)$ であるので、t_1 の処理が $pr(t_1)$ において終了した後に、同じプロセッサにおいて t の実行が開始される。また、t は t_2 の E先祖であるので、補題２より、t_2 の実行の開始時刻は t の実行の終了時刻よりも後である。従って、t_1 の処理が終了した後に、t_2 の実行が開始される。故に、$(t_1,t_2)\in D_{ipr}$ である。

以上により、常に補題が成立することが証明された。 ■

［定理１］ $(E\cup E_{ipr})^*\subseteq R_{ipr}$ である。

（証明） 補題１～５より明かである。 ■

定理１の逆（すなわち、$R_{ipr}\subseteq(E\cup E_{ipr})^*$）が成立するか否かは、現在までのところ証明できていない。しかしながら、G=(T,E) が待ち無し並列プログラムである場合は成立する。以下、その証明の概略を示す。。

［補題６］G=(T,E) を待ちなし並列プログラム、pr をプロセッサ割当関数、f を枝順序関数とする。 $t_1\neq t_2$、かつ、$(t_1,t_2)\in D_{ipr}$ であるならば、$(t_1,t_2)\in E_{ipr}$ である。（証明略） ■

［補題７］ G=(T,E) を並列プログラム、pr をプロセッサ割当関数、f を枝順序関数とする。 $R_{ipr}-D_{ipr}\subseteq E^*$ である。（証明略） ■

［定理２］ G=(T,E) を待ちなし並列プログラム、pr をプロセッサ割当関数、f を枝順序関数とする。 $(E\cup E_{ipr})^*=R_{ipr}$ である。

（証明） $R_{ipr}\subseteq(E\cup E_{ipr})^*$ であることを証明すればよい。補題７より、$R_{ipr}\subseteq E^*\cup D_{ipr}$ である。また、補題６より、$D_{ipr}\subseteq(E\cup E_{ipr})^*$ である。故に、$R_{ipr}\subseteq(E\cup E_{ipr})^*$ である。 ■

５．同期操作のオーバーヘッドの少ない並列プログラム

定理１より、与えられた先行制約 R に対して、$R\subseteq(E\cup E_{ipr})^*$ であれば、$R\subseteq R_{ipr}$であり、任意の実行例において R が満足されることが保証される。このことより、アルゴリズムFindMinPP_0 における R_{ipr} を、$(E\cup E_{ipr})^*$ で代用したアルゴリズムFindMinPP を考える。

［アルゴリズム FindMinPP(G=(T,E),pr,f)］

```
procedure FindMinPP(G=(T,E),pr,f);
  begin
  while (R⊆((E−{e})∪Eipr((T,E−{e}),pr,f))* であるような
          e∈E が存在する) do
    E:=E−{e};
  end;
```

■

［補題８］ $((t_1,t_3),(t_2,t_4))\in A_{ipr}$ ならば $depth(t_1:E)\leq depth(t_2:E)$ であ

る。また、$(t_1,t_2)\in E_{ipr}$ ならば $depth(t_1:E)\leq depth(t_2:E)$ である。
（証明） A_{ipr},E_{ipr} の定義において、各①～④の条件部で補題が成立することを仮定すれば、結論部で新たに A_{ipr},E_{ipr} の要素となるものに対しても補題が成立することを示すことにより、帰納的に証明を行う。

①の場合、$(t_1,t_2)\in E$ であるので、明らかに、$depth(t_1:E)<depth(t_2:E)$ であり、補題が成立する。

②の場合、$depth(t_1:E)\leq depth(t_1:E)$ であることより、明らかに補題が成立する。

③の場合、帰納法の仮定より、$depth(t_1:E)\leq depth(t_3:E)$ であり、補題が成立する。

④の場合、t_1 の E 親のうち E 深さが細大のものを u_1 とすると $depth(t_1:E)=depth(u_1:E)+1$ である。この u_1 に対しても A_{ipr},E_{ipr} の定義に述べたような u,t が存在する。(u,t) は (u_1,t_1) の A_{ipr} 子孫であるので、帰納法の仮定より、$depth(u_1:E)\leq depth(u:E)$ である。また、u は t の E 親であるので $depth(t:E)\geq depth(u:E)+1$ である。以上より $depth(t_1:E)\leq depth(t_2:E)$ であり、補題が成立する。
以上により、常に補題が成立することが証明された。 ■

与えられた G,pr,f に対して、E_{ipr},A_{ipr} を計算するには、始め、E_{ipr},A_{ipr} を共に空集合としておき、E_{ipr},A_{ipr} の定義により、新たに、E_{ipr},A_{ipr} の要素となるべきものを探し、それぞれの要素として加えることを繰り返せばよい。

ある時点において、$((t_1,t_3),(t_2,t_4))\in A_{ipr}$ であるか否かを判定するには、E_{ipr},A_{ipr} の定義の①，②，③より分かるように、その時点までに、t_2 に入る E_{ipr} 枝がすべて計算されていればよい。　またある時点において、$(t_1,t_2)\in E_{ipr}$ であるか否かを判定するには、E_{ipr},A_{ipr} の定義の④と補題８より、この時点において、$depth(u:E)<depth(t_2:E)$ であるような各タスク u から出る E 枝に入る A_{ipr} 枝が分かっていればよい。

従って、次のアルゴリズムは E_{ipr},A_{ipr} を計算する。

[アルゴリズム Make_E_{ipr}(G,pr,f)]

```
procedure Make_Eipr(G,pr,f);
  begin
  Aipr:=φ;
  Eipr:=φ;
  各 t2∈T に対して depth(t2:E) の小さい順に
    begin
    (t1≠t2) かつ (t1≠ts) かつ (depth(t1:E)≤depth(t2:E)) である各 t1
     に対して
      begin
```

```
    t1 の各 E 親 u1 に対して
        1) t≠t1、
        2) (u,t)∈E、
        3) (u,t) は (u1,t1) の Aipr(G=(T,E),pr,f)子孫、
        4) pr(t)=pr(t1)、かつ、
        5) t は t2 の E先祖
      であるような u,t が存在するならば
        Eipr:=Eipr ∪ {(t1,t2)}
    end;
  t2 の各 E 親 t1 に対して
    t2 の各 E 子 t3 に対して
      Aipr:=Aipr ∪ {((t1,t2),(t2,t3))};
      (* Aipr,Eipr の定義の①に対応*)
  t2 の各 E 子 t3 に対して
    t2 の各 E 子 t4 に対して
      f(t2,t3)<f(t2,t4) ならば
        Aipr:=Aipr ∪ {((t2,t3),(t2,t4))};
        (* Aipr,Eipr の定義の②に対応*)
  t2 の各 Eipr 親 t1 に対して
    t1 の各 E 子 t3 に対して
      t2 の各 E 子 t4 に対して
        Aipr:=Aipr ∪ {((t1,t3),(t2,t4))};
        (* Aipr,Eipr の定義の③に対応*)
  end;
end;                                                    ■
```

図２に、図１に示した並列プログラム G=(T,E) に FindMinPP を適用した結果を示す。ただし、R=E* としている。

６．むすび

プロセッサ割当 pr と枝順序関数 f が与えられた並列プログラム G=(T,E) を、FCFS と呼ばれる実行機構で実行する場合に、誘導される先行関係 Ripr(G,pr,f) の定義を行った。そして、タスク間に先行制約 R が与えられているとき、R を満足するためには R⊆Ripr(G,pr,f) であればよいことを示した。Ripr(G,pr,f) は意味の上から定義されたものであり、まだ、計算する方法が知られていない。ここでは、グラフ論的に定義することのできる枝集合 Eipr(G,pr,f) を用いることにより、一般の並列プログラムでは (E∪Eipr(G,pr,f))*⊆Ripr(G,pr,f) であり、待なし並列プログラムでは (E∪Eipr(G,pr,f))*⊆Ripr(G,pr,f) で

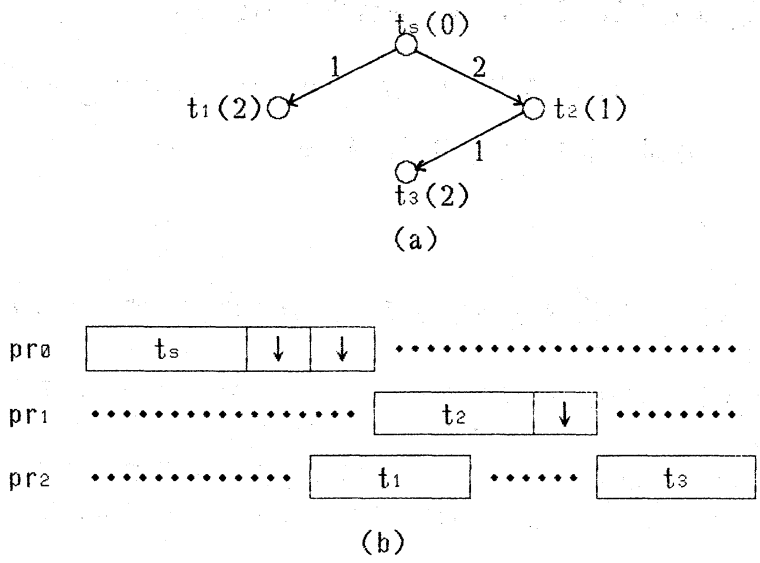

図２　FindMinPP の適用例

あることを証明した。従って、$R \subseteq (E \cup E_{ipr}(G,pr,f))^*$ であれば、先行制約 R は満足されることになる。これらのことより、$R \subseteq (E \cup E_{ipr}(G,pr,f))^*$ である限り不要な枝を削除すれば、R を満足し、かつ、同期の個数（枝の個数）が極小な並列プログラムが得られることを示した。

文　　献

1) 有田 ：FIFOキューを同期手段とする並列プログラムについて（Ⅰ）－待ちなし並列プログラム－，情報処理学会論文誌，Vol.24，No.2，pp.221-229 (1983).

2) 有田，荒木 ：FIFOキューを同期手段とする並列プログラムについて（Ⅱ）－ 並列プログラムの待ちなし変換－，情報処理学会論文誌，Vol.24，No.2，pp.230-237 (1983).

3) 有田，末吉 ：FIFOキューを同期手段とする並列プログラムについて（Ⅲ）－ 実行管理機構－，情報処理学会論文誌，Vol.24，No.6，pp.838-846 (1983).