共有メモリマルチプロセッサシステムにおける同期時間最適な無待機時計合せプロトコル

守屋　宣[†]　井上美智子[†]　増澤　利光[†]　藤原　秀雄[†]

Optimal Wait-Free Clock Synchronization Protocol on a Shared-Memory Multi-Processor System

Sen MORIYA[†], Michiko INOUE[†], Toshimitsu MASUZAWA[†], and Hideo FUJIWARA[†]

あらまし　共有メモリマルチプロセッサシステム，特に，システム内のすべてのプロセッサが大域パルスを共有するフェーズ内システムにおける故障耐性をもつ時計合せプロトコルを考察する．フェーズ内システムでは，正常なプロセッサはパルス発生時に同期して動作を行う．フェーズ内システムにおいて，パルス発生時にプロセッサが動作しないような故障を居眠り故障と呼ぶ．居眠り故障の起こるフェーズ内システムにおいて，同期時間と呼ばれるある特定パルス以上正常に動作し続けているすべてのプロセッサ同士の局所時計の時刻を一致させるプロトコルを無待機時計合せプロトコルと呼ぶ．これまで，フェーズ内システムにおける無待機時計合せプロトコルとして，同期時間 $4n^2-3n-1$ のプロトコルが提案されていた（n:プロセッサ数）．本論文では，同期時間 $12n$ の無待機時計合せプロトコルを提案する．また，無待機時計合せプロトコルの同期時間の下界が $n-1$ であることを証明し，本論文で提案するプロトコルが同期時間に関してオーダ的に最適であることを示す．

キーワード　共有メモリマルチプロセッサシステム，時計合せプロトコル，故障耐性，無待機性，同期時間

1. まえがき

マルチプロセッサシステムにおける重要な問題の一つに，プロセッサ間の同期を実現することがある．プロセッサ間の同期をとるための手段として，大域時計がよく用いられる．しかし，全プロセッサが共通の一つの時計を参照する方法では，その時計が故障すればどのプロセッサも大域時計を利用できなくなるという欠点があり，システム全体の信頼性は低い．そこで，各プロセッサが個別に時計を実現し，これらの時計を同期させるという方法が提案されている [1]〜[4]．この方法では，各プロセッサが個別に時計を実現するため，一部のプロセッサが故障しても正常なプロセッサ間で時計を同期するように実現するといった故障耐性をもたせ，信頼性を上げるということが考えられる．このような状況で，各プロセッサが管理する局所時計を同期させるプロトコルを時計合せプロトコルと呼ぶ．

故障プロセッサが存在するシステムにおける時計合せ問題は多くのアプリケーションにとって重要であり，またそれ自体も興味深い問題である．時計合せ問題は，これまで様々なモデルのもとで，様々な結果が示されている（[1], [2] など）．本論文では，各プロセッサが共通の大域パルスを共有するフェーズ内システム（in-phase system）（図 1）における時計合せプロトコルを考察する．故障プロセッサが存在するフェーズ内システムのもとでの時計合せプロトコルは，Dolev らによって提案された [3]．Dolev らは，プロセッサの故障として任意時間動作を停止した後に動作停止に気づかずに動作を再開するという居眠り故障（napping fault）を対象としている．彼らのプロトコルは，ある定数 k に対し，以下の二つの条件を保証する．(1) k パルスの間，正常に動作し続けたプロセッサは，それ以降正常に動作し続ける限り，局所時計の値は各パルスで 1 ずつ増える．(2) k パルスの間，正常に動作し続けた二つのプロセッサの局所時計の値は一致する．このプロトコルは，各プロセッサは少なくとも k パルス動作し続ければ他のプロセッサの動作にかかわらず局所時計の値を一致させることができるという意味で，無待機時計合せプロトコル（wait-free clock

[†] 奈良先端科学技術大学院大学情報科学研究科，生駒市
Graduate School of Information Science, Nara Institute of Science and Technology, Ikoma-shi, 630–0101 Japan

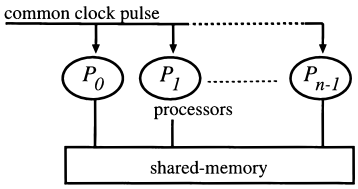

図 1　フェーズ内システム
Fig. 1　In-phase system.

表 1　フェーズ内システムにおける無待機時計合せプロトコル
Table 1　Wait-free clock synchronization protocols on an in-phase system.

	同期時間	
	上界	下界
Dolev ら [3]	$16n^2+n-1$	
Papatriantafilou ら [4]⋆	$4n^2-3n-1$	
本論文	$12n$	$n-1$

⋆ 自己安定無待機時計合せプロトコル

synchronization protocol）と呼ばれる．ここで，定数 k を同期時間（synchronization time）と呼ぶ．無待機時計合せプロトコルでは，居眠り故障を起こすプロセッサも k パルス正常に動作し続ければ局所時計を同期させることができ，また，任意個のプロセッサが居眠り故障を起こす場合でも故障プロセッサの動作に影響されることなく局所時計を同期させることができる．

過去に提案されたフェーズ内システムにおける無待機時計合せプロトコルを表 1 に示す．Dolev らは，同期時間 $16n^2+n-1$（n:プロセッサ数）の無待機時計合せプロトコルを提案した [3]．また，任意の初期システム状況から開始する実行においても大域時計を実現する自己安定無待機時計合せプロトコルとして，Dolev らは同期時間 $8n^3+n-1$ の自己安定無待機時計合せプロトコルを提案した [3]．また，Papatriantafilou らは，同期時間 $4n^2-3n-1$ の自己安定無待機時計合せプロトコルを提案した [4]．本論文では，同期時間 $12n$ の無待機時計合せプロトコル WCS の提案をし，プロトコル WCS の同期時間がオーダ的に最適であることを示す．

2. 諸　定　義

本章でシステムのモデル，無待機時計合せ問題を定義する．本論文で用いるシステムのモデル，無待機時計合せ問題の定義は，文献 [3], [4] のものと同じである．

n 個のプロセッサからなる共有メモリマルチプロセッサシステムを考える．プロセッサは共有変数を介してのみ通信を行うことができる．共有メモリマルチプロセッサシステム $\mathcal{S}$ を $\mathcal{S}=(\mathcal{P},\mathcal{V})$ で定義する．ここで，$\mathcal{P}$ はプロセッサの集合，$\mathcal{V}$ は共有変数の集合である．プロセッサは 0 から $n-1$ までの相異なる識別子をもつとし，識別子 i のプロセッサを P_i と表す．また，各共有変数は，所有者と呼ばれるある 1 プロセッサのみが書込みをでき，すべてのプロセッサが読出しをできるとする．各プロセッサは状態機械としてモデル化される．状態 s のプロセッサ P_i は，以下の (1) から (2) の動作を 1 ステップの操作として行い，次の状態 s' に遷移する．(1) 状態 s により決定するプロセッサ P_j が所有するすべての共有変数を読み出す(注 1)．(2) 状態 s と P_j が所有する共有変数の値をもとに，P_i が所有する共有変数を更新し，状態 s' に遷移する．

本論文では，フェーズ内システムと呼ばれる同期式マルチプロセッサシステムを扱う．フェーズ内システムとは，全プロセッサが共通の大域パルスを共有するシステムであり，全プロセッサは大域パルスを受けたときに 1 ステップの動作を同期して行う．ただし，本論文では後で述べる居眠り故障を考慮し，各パルスで全プロセッサが動作するとは限らないとする．システム $\mathcal{S}$ の大域状況を，全プロセッサと全共有変数の状態の組で表す．以降，このシステムの大域状況のことを，単に状況という．システムの実行は，状況と各パルスで動作したプロセッサの集合の無限または有限交代列 $E=c_0\pi_1c_1\cdots$ で表すことができる．ここで，各 c_t は状況を表し，特に c_0 は各プロセッサ，共有変数の特定の初期状態からなる初期状況を表す．また，π_t は t 番目のパルス発生時に 1 ステップの動作したプロセッサの集合を表す．実行 E は，有限であるときはある状況 c_{end} で終わる．この実行 E は，各 t に対し，π_t に属する各プロセッサが c_{t-1} をもとに同期して 1 ステップの動作を行い，状況が c_t になったことを意味する．また，t 番目のパルスが発生した（大域）時刻を時刻 t と呼ぶ．また，状況 c_t での変数 $P_i.x$ の値を $P_i.x(t)$ で表す．

プロセッサ P_i について，$P_i\notin\pi_t$ のとき，P_i は

(注 1)：実際的には，各プロセッサに対し一つの共有メモリセルが割り当てられており，各プロセッサは自分のセルを複数のフィールドに分割して使用する．プロセッサは各共有変数をフィールドに割り当てることにより，所有するすべての共有変数が一つのセルに格納できるとする．プロセッサ P_i は，状態 s により決定するプロセッサ P_j のセルの読出しを行うことにより，P_j が所有するすべての共有変数を読み出すことができる．

時刻 t で居眠り故障した，または単に居眠りしたという．P_i が時刻 t で居眠りしたならば，P_i は時刻 t では全く動作をしない．すなわち，c_{t-1} と c_t における P_i の状態と P_i が所有する共有変数の値は変わらない．システム $\mathcal{S}$ の実行 $E = c_0\pi_1c_1\cdots$ に対して，プロセッサ P_i が時刻 t まで連続して動作したパルス数を $work(P_i,t)$ と表す．すなわち，$work(P_i,t) = \max\{l|P_i \in \bigcap_{t-(l-1)\leqq t'\leqq t}\pi_{t'}\}$ であり，$work(P_i,t) = u$ のとき P_i が区間 $[t-u+1,t]$ で居眠りすることなく動作し続けたことを意味する．ただし，$P_i \notin \pi_t$ のときは，$work(P_i,t) = 0$ と定義する．

次に，フェーズ内システムにおける無待機時計合せ問題を定義する．各プロセッサ P_i は，P_i の局所時計を表す共有変数 $Clock$ をもつとする．任意の時刻 t に対し，t まで連続して動作したパルス数がある定数以上であるような任意のプロセッサ P_i, P_j の局所時計の値が一致し，以降 P_i が動作し続ける限り局所時計の値が 1 パルスごとに 1 ずつ増えることを保証するプロトコルを無待機時計合せプロトコルという．

［定義 1］ ある正整数 k に対し，任意の実行が次の二つの条件を満たすようなプロトコルを同期時間 k の無待機時計合せプロトコルという．

(i) 調整完了性（Adjustment）：任意の $t \geqq 1$, 任意のプロセッサ P_i に対し，$work(P_i,t) \geqq k+1$ ならば，$P_i.Clock(t) = P_i.Clock(t-1)+1$ が成り立つ．

(ii) 一致性（Agreement）：任意の $t \geqq 1$, 任意のプロセッサ P_i, P_j に対し，$work(P_i,t) \geqq k, work(P_j,t) \geqq k$ ならば，$P_i.Clock(t) = P_j.Clock(t)$ が成り立つ． □

3. 無待機時計合せプロトコル

本章で，フェーズ内システムにおける同期時間 $12n$ の無待機時計合せプロトコル *WCS*を提案する．プロセッサ P_i のプログラムを図 2 に示す．図 2 では，共有変数を *Clock*のように大文字で始まる変数名で表す．また，各プロセッサの状態を局所変数の集合で表し，局所変数を *cur*のように小文字で始まる変数名で表す．特に変数 *cur*は，共有変数を読み出すプロセッサの識別子を表す変数である．

3.1 プロトコル *WCS*

3.1.1 全体の流れ

本プロトコルでは，各プロセッサ P_i は以下の二つのモードをもつ．

- 調整中モード（*adjusting* mode）
- 調整済モード（*adjusted* mode）

また，調整中モードは三つのピリオド，準備ピリオド（$4n$ ステップ），調査ピリオド（n ステップ），調整ピリオド（n ステップ）から構成される．初期状況では，全プロセッサが調整中モードの準備ピリオドにいる．調整中モードにあるプロセッサ P_i は，手続き *adjust* に従って時計調整を行っていき，やがて調査ピリオド，調整ピリオドへ移行する．調整ピリオドにおいて手続き *check_adjusted*により時計調整が終了したと判定したならば，調整済モードへ移行する．調整済モードにあるプロセッサは，ステップごとに局所時計の値を 1 ずつ増やす．また，プロセッサ P_i はすべてのステップにおいて P_i 自身及び他のプロセッサの居眠りのチェックを行う（手続き *check_nap*）．時計調整を始めてから居眠りをしたプロセッサは，調整中モードにおける時計調整を正しくできない，または，調整済モードにおいて正しい時計の値を維持できないことになる．そこで，自分の居眠りを検知した場合，プロセッサ P_i は手続き *partial_reset*を行い，準備ピリオドの始めから時計調整をやり直す．また，他のプロセッサ P_j の居眠りを検知したならば，共有変数を通して P_j に知らせる．

本プロトコルにおける時計調整のアイデアを図 3 に示す．プロセッサ P_i が調整中モードで時計調整を開始してから次に故障を検知して *partial_reset*を行うまでの区間を世代と呼ぶ．各プロセッサは，現在の世代でのステップ数を表す共有変数 *Work_count* を使う．時計調整を開始したプロセッサは，現在の世代を自分より早く始めたプロセッサを *Work_count*により調べる．図 2 に示すように，現在の世代を後から始めたプロセッサは，自分より早く始めたプロセッサの局所時計だけをもとにして時計調整を行う．

以下，調整中モードにおける手続き *adjust* による時計調整の詳細について説明し，次に手続き *check_nap* による居眠りのチェックについて説明する．

3.1.2 時計調整（手続き *adjust*）

調整中モードにいるプロセッサは，準備ピリオド，調査ピリオドでは共有変数の読出しがその所有プロセッサの識別子が巡回する順序の繰返しになるようにステップを続け，調整ピリオドではプロトコルに従って読出しの順序を変更する．調査ピリオドで，他のプロセッサがそのときの世代を始めた時刻を調べる．調整ピリオドでは，自分より早く世代を始めた他のプロ

```
1   shared variables
2     Clock, 初期値 0
3     Count, 初期値 0
4     Work_count, 初期値 0
5     Gen, 初期値 1 /* 世代番号 */
6     Invalid_j(j = 0...n − 1), 初期値 0
7     Mode, "adjusting" or "adjusted", 初期値 "adjusting"
8     Sync, Pの部分集合, 初期値 ∅
9       /* 局所時計が一致するプロセッサの集合 */
10    Async, Pの部分集合, 初期値 ∅
11      /* 局所時計を無視, 棄却したプロセッサの集合 */
12  local variables
13    cur, 初期値 0
14    last_count_j(j = 0...n − 1), 初期値 0
15    last_gen_j(j = 0...n − 1), 初期値 0
16    last_my_count_j(j = 0...n − 1), 初期値 0
17    key_j(j = 0...n − 1)
18      /* 世代を始めた相対時刻 */
19    list[0...n − 1]
20      /* 世代を始めた順番 */
21    next, sync, pos

22  repeat forever do on receipt of a pulse
23    read P_cur's variables;
24    if (check_nap) then
25      partial_reset; /* 居眠りリセット */
26    elseif Mode = "adjusting" then adjust;
27    else /* Mode = "adjusted" */
28      Clock := Clock + 1;
29      next := cur + 1 (mod n);
30    last_count_cur := P_cur.Count;
31    last_gen_cur := P_cur.Gen,;
32    last_my_count_cur := Count;
33    cur := next;
34    Count := Count + 1;
35    Work_count := Work_count + 1;
36  end

37  procedure check_nap;
38    diff := (Count − last_my_count_cur)
39          −(P_cur.Count − last_count_cur);
40    if diff > 0 and P_cur.Gen = last_gen_cur
41      then Invalid_cur := P_cur.Gen;
42    if (Work_count ≧ n and diff < 0)
43      or (Work_count ≧ n and P_cur.Invalid_i = Gen)
44      then return true;
45    else return false;
46  end procedure;

47  procedure partial_reset;
48    next := 0; Mode := "adjusting";
49    Gen := Gen + 1;
50    Work_count := −1; /* becomes 0 at line 35*/
51  end procedure;

52  procedure adjust;
53    if 0 ≦ Work_count ≦ 4n − 1
54      then next := cur + 1(mod n);
55    elseif 4n ≦ Work_count ≦ 5n − 1 then
56      begin
57        if Invalid_cur < P_cur.Gen then
58          key_cur := P_cur.Work_count − Work_count
59        else key_cur := −1;
60        next := cur + 1(mod n);
61        if Work_count = 5n − 1 then sort_list
62      end
63    else /* Work_count ≧ 5n */
64      begin
65        if Invalid_cur = P_cur.Gen
66          or P_cur.Work_count < Work_count then
67          begin
68            Async := Async ∪ {P_cur};    /* 無視 */
69            pos := pos + 1; check_adjusted;
70            Clock := Clock + 1;
71          end
72        elseif P_cur.Mode = "adjusted" then
73          begin
74            if (Sync ⊄ P_cur.Async and
75              P_cur.Sync ≠ ∅ and Clock ≠ P_cur.Clock)
76              then partial_reset; /* 居眠りリセット */
77            else
78              begin
79                Clock := P_cur.Clock + 1;
80                if Sync ⊂ P_cur.Async
81                  then Async := Async ∪ Sync; /* 棄却 */
82                       Sync := ∅;
83                Sync := Sync ∪ {P_cur};
84                pos := pos + 1; check_adjusted;
85              end
86          end
87        if Work_count = 6n − 1
88          and Mode = "adjusting"
89          then partial_reset; /* 時間超過リセット */
90      end
91  end procedure;

92  procedure sort_list;
93    list[0..n − 1] := sort ids in the descending
94              lexicographic order of (key_id, id);
95    Sync := ∅; Async := ∅;
96    pos := 0; check_adjusted;
97  end procedure;

98  procedure check_adjusted;
99    if list[pos] = i
100     then Mode := "adjusted"; next := 0;
101   else next := list[pos];
102 end procedure;
```

図 2　プロトコル *WCS*
Fig. 2　Protocol *WCS*.

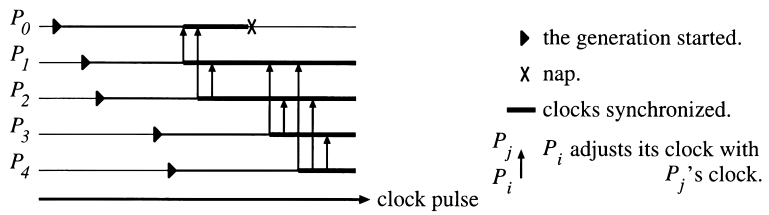

図 3　プロトコル *WCS*のアイデア
Fig. 3　Idea of protocol *WCS*.

セッサの局所時計と自分の局所時計を一致させる．また，$4n$ ステップの準備ピリオドは，調査ピリオドで他のプロセッサが時計調整を始めた時刻を正しく調べられることを保証するために必要とする（補題 1）．

調査ピリオドでは，プロセッサ P_iは $P_j.Work\_count$ により，P_i が世代を始めた時刻に対する，P_j がそのときの世代を始めた相対時刻を調べる．ただし，P_i が P_j の居眠りを検知したならば，P_j は遅くとも次に P_i の共有変数を読み出すステップで *partial_reset*を行うので，P_i は P_j を時計調整を遅く開始したプロセッサとして扱う．調整ピリオドでは，その所有者が時計調整を始めた時刻の順に共有変数を読み出す．その順序を知るため，調査ピリオドの最後のステップで手続き *sort_list*により P_i より早く時計調整を始めたプロセッサを時計調整を開始した時刻の順にソートし，その結果を配列 *list*に格納する．ただし，同時刻に時計調整を開始したプロセッサが複数存在するならば，識別子により順序を付ける．P_i より早く時計調整を始めたプロセッサがなければ，手続き *check_adjust*により，このステップで P_i は調整ピリオドに入ることなく調整済モードへ移行する．

調整ピリオドでは，P_i はまず最も早く時計調整を始めたプロセッサ，すなわち *list*の先頭にあるプロセッサの局所時計に自分の時計を合わせる．その後，原則として *list*の順に，P_i より早く時計調整を始めた他のプロセッサの局所時計と自分の局所時計が一致していることを確認していく．調整ピリオドのステップは以下のように行う．以下では，そのステップで読み出す共有変数の所有者を P_j とする．

- P_i が P_j の居眠りを検知したとき，または P_j が自身の居眠りを検知したときは，P_j の局所時計に合わせることも，一致確認もしない．次のステップでは *list*の次のプロセッサの共有変数を読み出す．このとき，P_i は P_j の時計を無視すると呼ぶ（68 行）．
- P_j がまだ調整中モードのときは，今回のステップでは P_j の局所時計に合わせない，または 一致確認をしない．次のステップでも再び P_j の読出しをする．
- P_i がこれまで局所時計を合わせた，または一致確認したプロセッサが信頼できないと判断したら，現在の局所時計を棄却し（81 行），P_j の局所時計に合わせる．このために，局所時計を合わせた，または一致確認したプロセッサ集合を表す $Sync$, 無視または棄却をしたプロセッサの集合 $Async$ を使う．
- P_i と P_j が，現在の世代で，共通のプロセッサ P_m の局所時計に合わせた，または一致確認しているにもかかわらず，両者の局所時計が一致しないならば，P_i は *partial_reset*を行い時計調整をやり直す．

プロセッサ P_i は，P_i より早く時計調整を始めたすべてのプロセッサ，すなわち配列 *list*において P_i より前にあるすべてのプロセッサの局所時計に対し，合わせる，一致確認をする，または無視したとき，手続き *check_adjust*により P_i は調整中モードから調整済モードに移行する．プロトコル *WCS*は，現在の世代の時計調整を始めてから P_i が居眠りすることなく動作し続けている場合は，$P_i.Work\_count = 6n$ に達する前に調整中モードから調整済モードへ移行することを保証する（補題 4 で証明）．そこで，調整済モードへ移行することなく $P_i.Work\_count = 6n$ に達するならば，P_i が居眠りしていたことになり，*partial_reset* を行い時計調整をやり直す．以降，この場合の *partial_reset* を時間超過リセット，他の場合，すなわち先に述べた場合と後で述べる手続き *check_nap* による場合を居眠りリセットと呼んで区別する．

3.1.3 居眠りのチェック（手続き *check_nap*）

各プロセッサは，初期状態からのステップ数を表す共有変数 *Count*（初期値 0）を使う．また，新しい世代の時計調整を開始するたびに更新する共有変数 *Gen*（初期値 1）をもつ．$Gen = g$ の間の時計調整を，世代 g の時計調整と呼ぶ．

プロセッサ P_i が P_j の共有変数を読み出すステップにおいて，前回の P_j の共有変数を読み出したステップからの $P_i.Count$, $P_j.Count$ の増分の差（図 2，第 38 行，$diff$ で表す）により，P_i は自身または P_j が居眠りしていたことを判定する．P_i が P_j の居眠りを検知したならば（第 40 行の条件式が成立），$P_j.Gen$ の値を P_i の変数 $Invalid_j$ に代入することにより，P_i は P_j の居眠りを P_j に知らせる．P_i は自身の居眠りを検知したときに $P_i.Work\_count \geqq n$ ならば（第 42 行または第 43 行の条件式が成立），*partial_reset*を行う(注 2)．プロセッサ P_i が居眠りをしたならば，複数のプロセッサとの *Count*の増分差の比較において P_i が P_i 自身の居眠りを検知し得る．このような居眠り検知の重複を避けるため，$P_i.Work\_count \geqq n$ を条

(注 2)：手続き check_nap は完全に居眠りを検知できるわけではない．例えば，すべてのプロセッサが同時に同期間居眠りをするような場合は，どのプロセッサも居眠りをしなかったかのように動作を継続する．しかし，プロトコル WCS が正当であるために十分な居眠り検出は行っている．

件としている．

3.2 *WCS* の正当性

プロトコル *WCS*が同期時間 $12n$ の無待機時計合せプロトコルであることを示す．本論文では，一部の補題に関しては証明を省略する．詳細は文献 [5] を参照されたい．

同期時間を，プロセッサが居眠りをしてから *partial_reset* を行うまでのステップ数と，時計調整を始めてから（初期状態または *partial_reset*を行ってから）調整済モードへ移行するまでのステップ数に分けて考える．プロトコル *WCS*では，調整ピリオドでの n 回のステップ後も調整中モードから調整済モードへ移行しないならば，$P_i.Work\_count(t'-1) = 6n-1$ なる時刻 t' で時間超過リセットを行う．補題 4 では，調整中モードのプロセッサが時計調整を始めてから居眠りをしていない，かつ居眠りリセットを行わないならば，$P_i.Work\_count(t'-1) = 6n-1$ なる時刻 t' までに調整済モードへ移行することを示す．補題 7 では，調整済モードの二つのプロセッサが十分長く居眠りすることなく動作し続けているならば，両者の局所時計の値が一致することを示す．補題 8 では，プロセッサが居眠りをしてから *partial_reset*を行うまでのステップ数がたかだか $6n$ であることを示す．補題 9, 10で，任意の実行に対し調整完了性と一致性が成り立つことを示す．

時計調整を始めた時刻の前後関係を表すため，$t_i < t_j$ または $t_i = t_j \wedge i > j$ ならば，$(t_i, i) \prec (t_j, j)$ と定義する．また，$(t_i, i) \prec (t_j, j)$ または $(t_i, i) = (t_j, j)$ ならば，$(t_i, i) \preceq (t_j, j)$ とする．プロセッサ P_i の時刻 t_i の手続き *sort_list*において，$list[p] = j$,$list[p'] = j'$ なるプロセッサ P_j,$P_{j'}$ に対し $p < p'$ が成り立つならば，$P_j \longrightarrow^i_{t_i} P_{j'}$ と記す．

手続き *sort_list*のソートの結果に関して以下の補題が成り立つ．

［補題 1］ 任意の時刻を t とする．区間 $[t+2n+1, t+5n]$ において，三つのプロセッサ $P_{j_1}, P_{j_2}, P_{j_3}$ は正常に動作し続け，かつ *partial_reset*を行わないとする．また，$(t_1, j_1) \preceq (t_2, j_2) \prec (t_3, j_3)$, $t_1 \geqq t + 4n + 1$ かつ $t_3 \leqq t + 5n$ である時刻 t_1, t_2, t_3 で，それぞれ $P_{j_1}, P_{j_2}, P_{j_3}$ が手続き *sort_list*を行ったとする．このとき，$P_{j_2} \longrightarrow^{j_3}_{t_3} P_{j_3}$ ならば，$P_{j'} \longrightarrow^{j_1}_{t_1} P_{j_1}$ なるすべての j' に対し，$P_{j'} \longrightarrow^{j_3}_{t_3} P_{j_2}$ または $P_{j_3} \longrightarrow^{j_3}_{t_3} P_{j'}$ が成り立つ． □

$steps(P_i, t', t)$ を以下のように定義する．$t' < t$ のとき，区間 $[t', t-1]$ でのステップ数とする．また，$t' = t$ のときは $steps(P_i, t', t) = 0$ とし，$t' > t$ のときは $-steps(P_i, t, t')$ と定義する．この定義より，任意の t_1, t_2, t_3 に対して $steps(P_i, t_1, t_2) + steps(P_i, t_2, t_3) = steps(P_i, t_1, t_3)$ が成り立つ．プロトコルより，$steps(P_i, t', t)$ に対し，以下の事実が成り立つ．

［事実 2］ 任意のプロセッサ P_i, P_j と時刻 t に対し，$steps(P_i, t', t)$ が以下を満たすような時刻 t' が存在するならば，P_i は区間 $[t', t-1]$ のある時刻で P_j が所有する共有変数を読み出すステップを行っている．

(a) $n \leqq P_i.Work\_count(t-1) \leqq 5n$ または $7n \leqq P_i.Work\_count(t-1)$ ならば，$steps(P_i, t', t) = n$

(b) $5n < P_i.Work\_count(t-1) < 7n$ ならば，$steps(P_i, t', t) = 2n$

(c) $P_i.Work\_count(t-1) < n$ ならば，$steps(P_i, t', t) = 3n-1$ □

P_i が世代 g の調整ピリオドでのステップ数，すなわち，時刻 t で *sort_list*を行った後，*partial_reset*を実行するか調整済モードへ移行する時刻 t' までのステップ数 $steps(P_i, t+1, t'+1)$ を $\alpha(P_i, g)$ と記す．

補題 3，4 では，以下に示す条件 A を満たすプロセッサ P_i を考える．

条件 A： P_i が時刻 t で *partial_reset*を実行し，$work(P_i, t+6n) \geqq 6n$ かつ $[t, t+6n]$ で居眠りリセットを行わない．

条件 A を満たす P_i に対し，$\bar{\alpha} = \alpha(P_i, P_i.Gen(t))$ とする．このとき，世代 $P_i.Gen(t)$ の時計調整で P_i が時間超過リセットを行うならば，時刻 $t+6n$ で *partial_reset*を行うので $\bar{\alpha} = n$ が成り立つ．よって，$\bar{\alpha} < n$ が成り立つならば，P_i が時刻 t までに調整済モードへ移行する．

条件 A を満たすプロセッサ P_i が調整ピリオドにいる区間 $[t+5n+1, t+5n+\bar{\alpha}]$ に属する各時刻 s に対し，2 種類のプロセッサ $reader_s, dest_s$ を以下のように時刻 $t+5n+\bar{\alpha}$ から再帰的に定義する（図 4）．まず，$reader_{t+5n+\bar{\alpha}} = P_i$ とする．時刻 s で $reader_s$ が読み出す共有変数の所有者を $dest_s$ と定義する．$reader_s, dest_s (t+5n+2 \leqq s \leqq t+5n+\bar{\alpha})$ に対し，$reader_s$ が時刻 s と $s-1$ で同じプロセッサの共有変数の読出しをするならば $reader_{s-1} = reader_s$ と定義し，異なる共有変数の読出しをするならば $reader_{s-1} = dest_s$ と定義する．ここで，$dest_s$ $(t+5n+1 \leqq s < t+5n+\bar{\alpha})$ のリスト

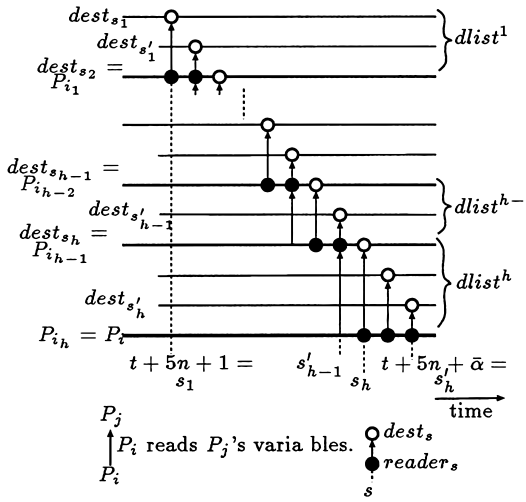

図 4　プロセッサ，時刻の定義
Fig. 4　Definitions of processors and times.

$dlist = dest_{t+5n+1}, dest_{t+5n+2}, \cdots, dest_{t+5n+\bar{\alpha}}$ を，$reader_s$ が同一でありかつ連続するように分割した極大部分リスト $dlist^1, dlist^2, \cdots, dlist^h$ を考える．各 $x(1 \leqq x \leqq h)$ に対し，$dlist^x$ で共通の $reader_s$ を P_{i_x}, $dlist^x = dest_{s_x}, dest_{s_x+1}, \cdots, dest_{s'_x}$ とし，P_{i_x} が世代 $P_{i_x}.Gen(s'_x - 1)$ において $sort_list$を行うならば，その時刻を $sort\_time^x$ とする．ここで，以下の補題が成り立つ．

［補題 3］ すべての $x(1 \leqq x \leqq h)$ に対し，以下が成り立つ．

(a)　$dlist^x$ の要素はすべて異なる．

(b)　P_{i_x} は区間 $[t+2n+1, s'^x - 1]$ で居眠りをしない．

(c)　$x < h$ ならば，$(t+4n+1, 0) \preceq (sort\_time^x, i_x) \prec (sort\_time^{x+1}, i_{x+1})$

(証明)　すべての $x(1 \leqq x \leqq h)$ に対して条件が成り立つことを $x = h$ の場合から帰納的に示す．

まず，$x = h$ の場合に条件 (a)，(b) が成り立つことを示す．定義より，$P_{i_x} = P_i$ である．また，$work(P_i, t+6n) \geqq 6n$ より区間 $[t+2n+1, s'_h - 1]$ で居眠りしていない．P_i は $[s_h, s'_h]$ で調整ピリオドにあるので，この区間では $list$ の順に共有変数を読み出している．したがって，$dlist^h$ は $dest_{s_h}$ と $dest_{s_h} \longrightarrow^{i_h}_{sort\_time^h} P_m \longrightarrow^{i^h}_{sort\_time^h} P_i$ なるすべての P_m からなり，$dlist^h$ の要素はすべて異なる．

次に，$x+1, \cdots, h$ に対して条件 (a)，(b)，(c) が成り立つならば，x に対して条件 (a)，(b)，(c) が成り立つことを示す．まず，(b) が成り立つことを示す．定義より，$P_{i_x} = dest_{s_{x+1}}$ である．$P_{i_{x+1}}$ は時刻 s'_x，s_{x+1} で連続して P_{i_x} の共有変数を読み出す．ここで，$P_{i_{x+1}}$ は時刻 s'_x で調整ピリオドにいるので $P_{i_{x+1}}.Work\_count(s'_x - 1) \geqq 5n$ であり，$(sort\_time^{x+1}, i_{x+1}) \preceq (sort\_time^h, i_h)$ が成り立つので，$2n+1 \leqq P_{i_{x+1}}.Work\_count(t+3n) \leqq 3n$ が成り立つ．よって，事実 2 より区間 $[t+2n+1, t+3n]$ に $P_{i_{x+1}}$ が P_{i_x} の共有変数を読み出すステップが存在する．$[t+3n, s'_x - 1]$ で P_{i_x} が居眠りをしているならば，遅くとも時刻 s'_x までに $P_{i_{x+1}}$ が P_{i_x} の居眠りを検知するか，時刻 $s'_x - 1$ までに P_{i_x} が自身の居眠りに気づいて次世代の時計調整を始めている．また，$P_i.Work\_count(t+2n) = 2n$ が成り立つので，事実 2 より区間 $[t+n+1, t+2n]$ に P_i が P_{i_x} の共有変数を読み出すステップが存在する．よって，$[t+2n+1, t+3n-1]$ で P_{i_x} が居眠りをしているならば，P_i に必ず検知される．時刻 $s'_x - 1$ で P_{i_x} は調整ピリオドにあるので，$P_{i_x}.Work\_count(t+5n) \geqq n$ が成り立ち，事実 2 より $[t+3n+1, t+5n]$ で P_i の共有変数を通して P_{i_x} が自身の居眠りを検知する．よって，時刻 s_{x+1} では P_{i_x} 以外のプロセッサの共有変数を読み出すので，矛盾する．よって，P_{i_x} は区間 $[t+2n+1, s'_x - 1]$ で居眠りをしていない．すなわち，(b) が成り立つ．

また，時刻 $s'_x - 1$ で P_{i_x} は調整中モードにいることより $P_{i_x}.Work\_count(s'^x - 1) < 6n$ が成り立つので $sort\_time^x \geqq t+4n+1$ が成り立つ．区間 $[t+4n+1, sort\_time^{x+1}]$ で P_{i_x}, $P_{i_{x+1}}$ はともに居眠りしていないので，$P_{i_x} \longrightarrow^{i_{x+1}}_{sort\_time^{x+1}} P_{i_{x+1}}$ より，$(sort\_time^x, i_x) \prec (sort\_time^{x+1}, i_{x+1})$ が成り立つ．すなわち，(c) が成り立つ．

$(sort\_time^x, i_x) \preceq (sort\_time^h, i_h)$ が成り立つので，区間 $[s_x, s'_x]$ で P_{i_x} は調整ピリオドにある．よって，$dlist^x$ は $dest_{s_x}$ と $dest_{s_x} \longrightarrow^{i_x}_{sort\_time^x} P_m \longrightarrow^{i_x}_{sort\_time^x} dest_{s'_x}$ なるプロセッサ P_m からなり，$dlist^x$ の要素はすべて異なる．すなわち，(a) が成り立つ．　□

［補題 4］ 条件 A を満たすプロセッサ P_i は時刻 $t+6n$ までに調整済モードへ移行する．

(証明)　すべての s $(t+5n+1 \leqq s \leqq t+5n+\bar{\alpha})$ の間で $dest_s$ が異なり，かつ $dest_s$ として P_i が現れないことを示すことにより，$\bar{\alpha} < n$ が成り立つことを示す．

$h = 1$ の場合，すなわち $dlist = dlist^1$ の場合，補

題 3 (a) より $dlist$ の要素はすべて異なり，また $dlist$ に P_i が現れない．したがって，$\bar{\alpha} < n$ が成り立つ．

次に $h > 1$ の場合を考える．補題 3 (a) より，任意の $x(1 \leqq x \leqq h)$ に対し $dlist^x$ の要素がすべて異なる．以下では，任意の異なる $x, y(1 \leqq x \leqq h, 1 \leqq y \leqq h)$ に対し，P_i が $dlist^x$ に現れず，$dlist^x$ と $dlist^y$ に共通要素が現れないことを示す．補題 3 (b) より，任意の $x, y(1 \leqq x < y \leqq h)$ に対し，P_{i_x} は区間 $[t + 2n + 1, t + 5n]$ を含む区間 $[t + 2n + 1, s'_x - 1]$ で正常に動作し続けている．また，補題 3 (c) より，$(t+4n+1, 0) \preceq (sort\_time^x, i_x) \prec (sort\_time^y, i_y)$ が成り立つ．したがって，$P_{j_1} = P_{i_x}$, $P_{j_2} = P_{i_{y-1}}$, $P_{j_3} = P_{i_y}$, $t_1 = sort\_time^x$, $t_2 = sort\_time^{y-1}$, $t_3 = sort\_time^y$ と定めたときに補題 1 が成り立つ．すなわち，$P_{j'} \longrightarrow^{j_1}_{sort\_time^x} P_{j_1}$ なるプロセッサ $P_{j'}$ に対して，$P_{j'} \longrightarrow^{j_3}_{sort\_time^y} P_{j_2}$ または $P_{j_3} \longrightarrow^{j_3}_{sort\_time^y} P_{j'}$ が成り立つ．したがって，$dlist^x$ と $dlist^y$ に共通要素が現れることはない．また，$P_i = P_{i_h}$ なので，補題 1 より P_i は $dlist^x$ に現れない． □

プロトコル *WCS*では，プロセッサはすべての居眠りを検知するわけではない．よって，二つのプロセッサ P_i,P_j が居眠りするタイミングにより，P_i は「P_j は P_i より後で時計調整を始めた」と判定するときに，P_j が「P_i は P_j より後で時計調整を始めた」と矛盾した判定をする場合がある．この場合，P_i, P_j は互いに局所時計を参照することなく調整済モードへ移行すると考えられる．しかし，手続き *sort_list*のソートの結果に関して以下の補題が成り立ち，P_i, P_j が十分長く動作しているならば両者が矛盾した判定をすることはない．

［補題 5］ 任意の時刻を t, 任意の異なるプロセッサを P_i, P_j とする．P_i, P_j はそれぞれ時刻 t_i, t_j で *sort_list*を行い，かつ，それぞれ区間 $[t_i, t], [t_j, t]$ で *partial_reset*を行わなかったとする．このとき，$work(P_i, t) > 4n$ かつ $work(P_j, t) > 4n$ ならば，$P_j \longrightarrow^i_{t_i} P_i$ または $P_i \longrightarrow^j_{t_j} P_j$ が成り立つ． □

手続き *adjust*の説明で述べたように，プロセッサ P_i が自分の局所時計をある時刻 $t_{i,j}$ でプロセッサ P_j の局所時計に合わせた，または P_j の局所時計と一致することを確認したあと，P_i が P_j の局所時計を棄却することがある．しかし，$t_{i,j}$ 以降に P_i, P_j がともに *partial_reset*をすることなく動作し続けるような場合には以下の補題が成り立つ．

［補題 6］ 任意の異なるプロセッサ P_i, P_j に対し，ある時刻 t で P_i, P_j がともに調整済モードであったとし，$work(P_i, t) > 4n, work(P_j, t) > 4n$ が成り立つとする．また，時刻 $t_{i,j}$ で P_i は自分の局所時計を P_j の局所時計に合わせた，または P_j の局所時計と一致することを確認したとし，区間 $[t_{i,j}, t]$ で P_i, P_j はともに *partial_reset*を行わないとする．このとき，区間 $[t_{i,j}, t]$ で P_i は P_j の局所時計を棄却しない． □

［補題 7］ 任意の時刻を t, 任意の異なるプロセッサを P_i, P_j とする．時刻 t で P_i, P_j がともに調整済モードであったとする．このとき，$work(P_i, t) > 4n, work(P_j, t) > 4n$ ならば，$P_i.Clock(t) = P_j.Clock(t)$ が成り立つ．

(証明) P_i, P_j は，それぞれ世代 $P_i.Gen(t), P_j.Gen(t)$ の時計調整において，*sort_list*を行っている．P_i, P_j が *sort_list*を行った時刻をそれぞれ t_i, t_j とすると補題 5 より $P_j \longrightarrow^i_{t_i} P_i$ または $P_i \longrightarrow^j_{t_j} P_j$ が成り立つ．一般性を失うことなく，$P_j \longrightarrow^i_{t_i} P_i$ と仮定する．このとき，P_i は世代 $P_i.Gen(t)$ のある時刻 t' で自分の局所時計を P_j の局所時計の値と合わせるか，一致していることを確認し，区間 $[t', t]$ で P_j は *partial_reset*を行わない．よって，$P_i.Clock(t') = P_j.Clock(t'-1)+1$ が成り立つ．ここで，補題 6 より，区間 $[t', t]$ で P_i が P_j の局所時計を棄却することはない．また，区間 $[t', t]$ で P_i, P_j はともに *partial_reset*をすることはないので，それぞれステップごとに局所時計の値を 1 ずつ増やす．したがって，

$$steps(P_i, t', t) = steps(P_j, t', t) \tag{1}$$

であることを示せばよい．ここで，$t' \geqq t - 4n + 1$ ならば，区間 $[t', t]$ では P_i, P_j はともに正常に動作し続けるので明らかに式 (1) は成り立つ．以下では，$t' < t - 4n + 1$ の場合を考える．

事実 2 より，区間 $[t - 4n + 1, t - n]$ のある時刻 t'' で P_i は P_j の局所変数を読出しするステップを行っており，また $t''(\leqq t - n)$ より後で P_j は P_i の局所変数を読出しするステップを行っている．区間 $[t', t]$ で P_i, P_j はともに *partial_reset*を行わないので，区間 $[t', t'']$ の各ステップで P_i が P_i または P_j の居眠りを検知することはない．したがって，$steps(P_i, t', t'') = steps(P_j, t', t'')$ である．更に，区間 $[t, '' t]$ では P_i, P_j はともに正常に動作し続けるので $steps(P_i, t, '' t) = steps(P_j, t, '' t)$ が成り立つ．したがって，式 (1) は成り立つ． □

次の補題で，居眠りをしてから *partial_reset*を行うまでのステップ数を考察する．

［補題 8］ プロセッサ P_i が時刻 t で *partial_reset*を実行したならば，$work(P_i,t) < 6n$ が成り立つ．

(証明) P_i は時刻 t で P_j が所有する共有変数を読み出すとし，t より前で最後に P_i が V_j を読み出すステップを行った時刻を t' とする．時刻 t の *partial_reset*が時間超過リセットならば，補題 4 より $work(P_i,t) < 6n$ が成り立つ．時刻 t の *partial_reset*が居眠りリセットならば，以下の (a)〜(c) の条件式のうちいずれかが成り立つ．

(a) 第 42 行：$P_i.Work\_count(t-1) \geq n$ かつ $P_i.diff(t) < 0$

(b) 第 43 行：$P_i.Work\_count(t-1) \geq n$ かつ $P_j.Invalid_i \geqq Gen$

(c) 第74–75行:$P_i.Sync(t-1) \not\subset P_j.Async\,(t-1)$, $P_j.Sync \neq \emptyset$ かつ $P_i.Clock(t-1) \neq P_j.Clock(t-1)$

(a)，(b) が成り立つ場合に $work(P_i,t) < 6n$ が成り立つことは，事実 2 を用いて示される．以下では，(c) が成り立つ場合を考察する．

時刻 t で第76行を行うことより $P_i.Work\_count(t-1) < 6n$ であり，$P_i.Sync(t-1) \neq \emptyset$ より P_i は $5n \leqq P_i.Work\_count(t_m-1) < P_i.Work\_count(t-1)$ である時刻 t_m で，調整済モードのプロセッサ P_m の局所時計と自分の局所時計を合わせるか，一致していることを確認している．このとき，$P_i.Clock(t_m) = P_m.Clock(t_m-1)+1$ が成り立つ．以下では，$work(P_i,t) \geqq 6n$ を仮定して矛盾を導く．

P_i, P_j が t 以前で最後に *sort_list*を行った時刻をそれぞれ t_i, t_j とする．また，P_i が t 以前で最後に *partial_reset* を行った時刻を t^0, すなわち $Work\_count(t^0) = 0$ とする．ただし，t 以前に *partial_reset* を行っていないならば，$t^0 = 0$ とする．$work(P_i,t) \geqq 6n$ かつ $P_i.Work\_count(t-1) < 6n$ より，P_i は $[t^0,t]$ では居眠りすることなく正常に動作し続ける．事実 2 より，$[t^0+1, t^0+n]$ に P_i が P_j, P_m それぞれの局所変数を読み出すステップが存在する．また，P_i は，t_m より前で P_m の，t より前で P_j の居眠りを検知しない．したがって，$work(P_m,t_m-1) \geq t_m - t^0 - n > 4n$, $work(P_j,t-1) > work(P_j,t_m-1) > 4n$ が成り立つ．以下，時刻 t_m-1 での P_j のモードにより場合分けして考える．

(c1) 時刻 t_m-1 で P_j が調整済モードの場合

補題 7 より $P_m.Clock(t_m-1) = P_j.Clock(t_m-1)$ が成り立つ．区間 $[t_m, t-1]$ で P_i, P_j が居眠りすることはないので，$P_i.Clock(t-1) = P_j.Clock(t-1)$ が成り立ち，(c) に矛盾する．

(c2) 時刻 t_m-1 で P_j が調整中モードの場合

P_i, P_j はともに $t_m - 3n (> t-4n)$ 以降の各プロセッサの *Work_count* の値をもとに *sort_list*を行う．P_i, P_j, P_m は $[t_m-3n, t_m-1]$ で正常に動作し続けるので，$P_m \longrightarrow^i_{t_i} P_j$ より $P_m \longrightarrow^j_{t_j} P_j$ が成り立つ．したがって，P_j は世代 $P_j.Gen(t-1)$ 中のある時刻 t'_m で P_m の局所時計に自分の局所時計を合わせるか，P_m と自分の局所時計が一致することを確認している．すなわち，$P_j.Clock(t'_m) = P_m.Clock(t'_m-1)+1$ が成り立つ．P_m は区間 $[t'_m, t_m-1]$ または $[t_m, t'_m-1]$ で正常に動作し続け，P_j は区間 $[t'_m, t-1]$ で正常に動作し続ける．したがって，$steps(P_j,t'_m,t) = steps(P_m,t'_m,t_m) + steps(P_i,t_m,t)$ より $P_i.Clock(t-1) = P_j.Clock(t-1)$ が成り立ち，(c) に矛盾する． □

［補題 9］（調整完了性） 任意の $t \geq 1$, 任意のプロセッサ P_i に対し，$work(P_i,t-1) \geqq 12n$ ならば $P_i.Mode(t-1) =$ "*adjusted*" が成り立ち，$work(P_i,t) > 12n$ ならば $P_i.Clock(t) = P_i.Clock(t-1)+1$ が成り立つ．

(証明) プロセッサ P_i が $t-12n$ 以前で最後に居眠りした時刻を t_n とすると，t_n 以降の時刻 t' で *partial_reset* を実行した場合，補題 8 より，$work(P_i,t') < 6n$ が成り立つ．よって，$work(P_i,t) \geqq 12n$ ならば，P_i は時刻 $t-6n$ 以降に *partial_reset*を行うことはない．よって，$P_i.Work\_count(t-1) \geqq 6n$ が成り立ち，補題 4 より $P_i.Mode(t-1) =$ "*adjusted*" が成り立つ．また，$work(P_i,t) > 12n$ ならば，P_i は時刻 t で調整済モードのステップを行うので $P_i.Clock(t) = P_i.Clock(t-1)+1$ が成り立つ． □

［補題 10］（一致性） 任意の $t \geqq 1$, 任意のプロセッサ P_i, P_j に対し，$work(P_i,t) \geqq 12n, work(P_j,t) \geqq 12n$ ならば $P_i.Clock(t) = P_j.Clock(t)$ が成り立つ．

(証明) 補題 9 より，$P_i.Mode(t-1) =$ "*adjusted*" かつ $P_j.Mode(t-1) =$ "*adjusted*" である．よって，補題 7 より $P_i.Clock(t) = P_j.Clock(t)$ が成り立つ．

□

［定理 11］ プロトコル WCS は，同期時間 $12n$ の無待機時計合せプロトコルである． □

また，同期時間に関して以下の定理が成り立ち，プロトコル WCSの同期時間はオーダ的に最適である．

［定理 12］ 同期時間が $n-2$ 以下の無待機時計合せプロトコルは存在しない．

(証明) 同期時間 $k \leqq n-2$ の時計合せプロトコル A が存在すると仮定し，矛盾を導く．次の三つの条件を満たす二つの実行 $E = c_0\pi_1c_1\cdots, E' = c'_0\pi'_1c'_1\cdots$ を考える．以下，$Clock_E$,$Clock_{E'}$ と記すときは，それぞれ実行 E,E' における $Clock$ の値を表す．

(条件 1) ある時刻 t に対し，$c_0\pi_1c_1\cdots c_t = c'_0\pi'_1c'_1\cdots c'_t$ が成り立ち，すべてのプロセッサ P に対して $work(P,t) \geqq k$ が成り立つ．

このとき，一致性より，$P_0.Clock_E(t) = \cdots = P_{n-1}.Clock_E(t) = P_0.Clock_{E'}(t) = \cdots = P_{n-1}.Clock_{E'}(t)$ が成立する．この値を $clock(t)$ とする．

(条件 2) 実行 E に対し，$\pi_{t+1} = \pi_{t+2} = \cdots = \pi_{t+n-2} = \{P_0\}$ が成り立つ．

このとき，調整完了性より，実行 E では $P_0.Clock_E(t+n-2) = clock(t)+n-2$ が成り立つ．ここで，P_0 の区間 $[t+1, t+n-2]$ でのステップ数は $n-2$ なので，その間にあるプロセッサが所有する共有変数の読出しをしない．そのプロセッサを $P_i(i \neq 0)$ とする．

(条件 3) 実行 E' に対し，$\pi'_{t+1} = \{P_i\}, \pi'_{t+2} = \cdots = \pi'_{t+n-1} = \{P_0, P_i\}$ が成り立つ．

このとき，調整完了性より，実行 E' において

$$P_i.Clock_{E'}(t+n-1) = clock(t)+n-1$$

が成り立つ．また，両実行 E, E' において P_0, P_i 以外のプロセッサは区間 $[t+1, t+n-2]$ ではステップしないので，実行 E' における区間 $[t+2, t+n-1]$ での P_0 の動作は，実行 E における区間 $[t+1, t+n-2]$ での P_0 の動作に一致する．よって，

$$\begin{aligned} P_0.Clock_{E'}(t+n-1) &= P_0.Clock_E(t+n-2) \\ &= clock(t)+n-2 \end{aligned}$$

が成り立つ．しかし，一致性より $P_i.Clock_{E'}(t+n-1) = P_0.Clock_{E'}(t+n-1)$ が成り立つはずであり，矛盾する． □

4. むすび

本論文では，共有メモリマルチプロセッサシステムにおける時計合せプロトコルについて考察した．フェーズ内システムにおける同期時間 $12n$ の無待機時計合せプロトコル WCS を提案し，このプロトコルが同期時間に関してオーダ的に最適であることを示した．

同期時間 $O(n^2)$ の Dolev ら，Papatriantafilou らのプロトコルでは 1 ステップの計算量が定数である．これに対し，プロトコル *WCS*は手続き *sort_list*を行うステップでプロセッサが要素数 n のソートを行うため，1 ステップで計算量 $O(n\log n)$ の内部計算が可能なシステムにしか適用できない．しかし，プロトコル WCS をヒープを使用するように修正すれば，1 ステップを計算量 $O(\log n)$ で行う同期時間 $12n$ のプロトコルを構成できる．

今後の課題としては，同期時間が $O(n)$ の自己安定性のある無待機時計合せプロトコルの考察などが挙げられる．

謝辞 本研究に関して多くの有益な助言をして頂いた井上智生助手をはじめとする本学情報論理学講座の皆様方に深く感謝致します．なお，本研究の一部は，文部省科学研究費補助金（特別領域研究 B-2 10205218, 奨励研究 A 09780279, 09780281）による．

文 献

[1] D. Dolev, J. Halpern, and H. Strong, “On the possibility and impossibility of achieving clock synchronization,” J. Comput. Syst. Sci., vol.32, no.2, pp.230–250, 1986.

[2] J. Halpern, B. Simons, R. Strong, and D. Dolev, “Fault-tolerant clock synchronization,” Proceedings of the 3rd ACM Symposium on Principles of Distributed Computing, pp.89–102, 1984.

[3] S. Dolev and J. Welch, “Wait-free clock synchronization,” Proceedings of the 12th ACM Symposium on Principles of Distributed Computing, pp.97–108, 1993.

[4] M. Papatriantafilou and P. Tsigas, “On self-stabilizing wait-free clock synchronization,” Proceedings of the 4th Scandinavian Workshop on Algorithm Theory(LNCS 824), pp.267–277, 1994.

[5] 守屋 宣，井上美智子，増澤利光，藤原秀雄，“共有メモリマルチプロセッサシステムにおける同期時間最適な無待機時計合せプロトコル，” 技術報告，NAIST Information Science Technical Report, NAIST-IS-TR99004, 1999. http://isw3.aist-nara.ac.jp/IS/TechReport2/report/99004.ps.

（平成 11 年 3 月 26 日受付，7 月 28 日再受付）

守屋　　宣　（学生員）

平 7 阪大・理・数学卒．平 9 奈良先端科学技術大大学院博士前期課程了．現在，同大博士後期課程に在学中．主に分散アルゴリズムに関する研究に従事．

井上美智子　（正員）

昭 62 阪大・基礎工・情報卒．平 1 同大大学院博士前期課程了．同年富士通研究所（株）入社．平 7 阪大大学院博士後期課程了．奈良先端科学技術大学院大学情報科学研究科助手，現在に至る．分散アルゴリズム，グラフ理論，テスト容易化設計，高位合成の研究に従事．工博．IEEE，情報処理学会，人工知能学会各会員．

増澤　利光　（正員）

昭 57 阪大・基礎工・情報卒．昭 62 同大大学院博士後期課程了．同年同大情報処理教育センター助手．同大基礎工助教授を経て，平 6 奈良先端科学技術大学院大学情報科学研究科助教授，現在に至る．平 5 コーネル大客員準教授（文部省在外研究員）．分散アルゴリズム，並列アルゴリズム，テスト容易化設計，テスト容易化高位合成に関する研究に従事．工博．ACM，IEEE，EATCS，情報処理学会各会員．

藤原　秀雄　（正員）

昭 44 阪大・工・電子卒．昭 46 同大大学院博士後期課程了．阪大工学部助手，明大理工学部教授を経て，現在奈良先端科学技術大学院大学情報科学研究科教授．昭 56 ウォータールー大客員助教授．昭 59 マッギル大客員準教授．論理設計，高信頼設計，設計自動化，テスト容易化設計，テスト生成，並列処理，計算複雑度に関する研究に従事．著書 “Logic Testing and Design for Testability”（The MIT Press）など．大川出版賞．工博．IEEE，情報処理学会各会員，IEEE Fellow，IEEE Golden Core Member．