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わせプロトコル

守屋 宣　井上美智子　増澤利光　藤原秀雄

奈良先端科学技術大学院大学 情報科学研究科
〒 630-0101 奈良県生駒市高山町 8916-5
e-mail:{sen-m, kounoe, masuzawa, fujiwara}@is.aist-nara.ac.jp

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持つ時計合わ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よぶ. 居眠り故障のおこるフェーズ内システムにおいて, 同期時間とよ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わせプロトコルとよぶ. これまで, フェーズ内システムにおける無待機時計合わ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 $12n$ の無待機時計合わせプロトコルを提案する. また, 無待機時計合わ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ー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, 時計合わせプロトコル, 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１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１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, 2, 3, 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持たせ, 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わせプロトコルとよぶ.

故障プロセッサが存在するシステムにおける時計合わ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. 時計合わせ問題は, これまで様々なモデルの下で, 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) における時計合わ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下での時計合わせプロトコルは, 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２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２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関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, 無待機時計合わせプロトコル (wait-free clock synchronization protocol) とよばれる. ここで, 定数 kを同期時間 (synchronization time) とよぶ. 無待機時計合わせプロトコルでは, 居眠り故障をおこすプロセッサも 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お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わせプロトコルを表 1に示す. Dolev らは, 同期時間 $16n^2+n-1$(n:プロセッサ数) の無待機時計合わ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わ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わ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わせプロトコルを提案した [4]. 本論文では, 同期時間 $12n$ の無待機時計合わせプロトコル WCSの提案をし, プロトコル WCSの同期時間がオーダー的に最適であることを示す.

2　諸定義

本論文では, n 個のプロセッサから成る共有メモリマルチプロセッサシステムを考える. プロセッサは複数の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持つとし, 識別子 i のプロセッサを P_iと表す. また, 各共有変数は, 所有者とよばれるある１プロセッサのみが書き込みをでき, すべてのプロセッサが読み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. (2) 状態 s と P_jが所有する共有変数の値を基に, 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

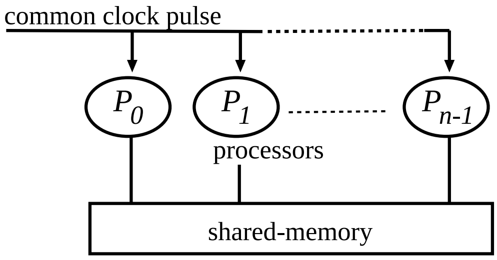

図 1: フェーズ内システム

表 1: フェーズ内システムにおける無待機時計合わせプロトコル

	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わせプロトコル

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}を基に同期して 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 とよぶ．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，$work(P_i,t) = \max\{l|P_i \in \bigcap_{t-(l-1)\le t'\le t}\pi_{t'}\}$ 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わせ問題を定義する．各プロセッサ P_iは，P_iの局所時計を表す共有変数 $Clock$を持つとする．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 パルス毎に 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わ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 2 つの条件を満たすようなプロトコルを同期時間 kの無待機時計合わせプロトコルという．

(i) 調整完了性 *(Adjustment)*： 任意の $t \ge 1$，任意のプロセッサ P_iに対し，$work(P_i,t) \ge k+1$ 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 $t \geq 1$, 任意のプロセッサ P_i, P_j に対し, $work(P_i, t) \geq k, work(P_j, t) \geq k$ ならば,

$$P_i.Clock(t) = P_j.Clock(t)$$

が成り立つ. ■

3 プロトコル

本節で, フェーズ内システムにおける同期時間 $12n$ の無待機時計合わせプロトコル WCS を提案する. プロセッサ P_i のプログラムを図3に示す. 図3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 概略

本プロトコルでは, 各プロセッサ P_i は2つのモード, 調整中モード ($adjusting$ mode), 調整済モード ($adjusted$ mode) を持つ. 初期状況では, 全プロセッサが調整中モードで時計調整を開始する. 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う. 調整中モードにおいて手続き $check\_adjusted$ 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, ステップ毎に局所時計の値を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 自身および他のプロセッサの居眠りのチェックを行う (手続き $check\_nap$). 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. 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, 時計調整を始めからやり直す.

本プロトコルにおける時計調整のアイデアを図2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よぶ. 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まず, 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基にして時計調整を行う.

以下, 手続き $check\_nap$ による居眠りのチェックについて説明し, 次に手続き $adjust$ による時計の調整の詳細について説明する.

3.1.2 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) を持つ. $Gen = g$ の間の時計調整を, 世代 g の時計調整とよぶ.

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3, 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 $P_i.Work\_count \geq n$ ならば (第42行または第43行の条件式が成立), $partial\_reset$ を行う. 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うる. このような居眠り検知の重複を避けるため, $P_i.Work\_count \geq n$ を条件としている.

ここで述べた以外に調整中モード中に $partial\_reset$ を行う場合がある. それらについては, 手続き $adjust$ の中で説明する.

3.1.3 時計の調整 (手続き $adjust$)

調整中モードにおける時計調整は, 3つのピリオドから構成される. 時刻 $t-1$ における $Work\_count$ の値に従って, 時刻 t にそれぞれ以下のピリオドのステップを行う.

- 準備ピリオド ($0 \leq Work\_count(t-1) \leq 4n-1$)
- 調査ピリオド ($4n \leq Work\_count(t-1) \leq 5n-1$)

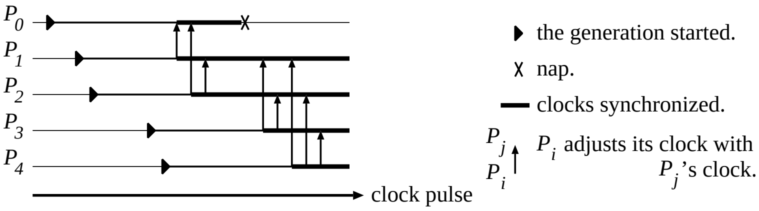

図 2: プロトコル WCSのアイデア

- 調整ピリオド $(5n \le Work\_count(t-1))$

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み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り返しになるようにステップを続け, 調査ピリオドではプロトコルに従って読み出しの順序を変更する. 調査ピリオドでは, 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). 準備ピリオドでは, 居眠りのチェックのみを行う.

調査ピリオドで, 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ない, または 一致確認をしない. 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よぶ (第 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の読み出しをする.

- 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 (第 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

```
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      $Sync$, $\mathcal{P}$の部分集合, 初期値$\emptyset$
9          /* 局所時計が一致するプロセッサの集合 */
10     $Async$, $\mathcal{P}$の部分集合, 初期値$\emptyset$
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 ($Work\_count \geq n$ and $diff < 0$)
43       or ($Work\_count \geq n$ 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 $0 \leq Work\_count \leq 4n-1$
54       then $next := cur + 1 (\textbf{mod}\ n)$;
55     elseif $4n \leq Work\_count \leq 5n-1$ 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         $next := cur + 1 (\textbf{mod}\ n)$;
61         if $Work\_count = 5n-1$ then $sort\_list$
62       end
63     else /* $Work\_count \geq 5n$ */
64       begin
65         if $Invalid_{cur} = P_{cur}.Gen$
66           or $P_{cur}.Work\_count < Work\_count$ then
67           begin
68             $Async := Async \cup \{P_{cur}\}$;    /* 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 ($Sync \not\subset P_{cur}.Async$ and
75               $P_{cur}.Sync \neq \emptyset$ and $Clock \neq P_{cur}.Clock$)
76               then $partial\_reset$; /* 居眠りリセット */
77             else
78               begin
79                 $Clock := P_{cur}.Clock + 1$;
80                 if $Sync \subset P_{cur}.Async$
81                   then $Async := Async \cup Sync$; /* 棄却 */
82                         $Sync := \emptyset$;
83                 $Sync := Sync \cup \{P_{cur}\}$;
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     $Sync := \emptyset$; $Async := \emptyset$;
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 3: プロトコル *WCS*

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, P_iが居眠りしたことにより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, $partial\_reset$ を行い時計調整をやり直す. この場合の $partial\_reset$ を時間超過リセット, 他の場合の $partial_reset$を居眠りリセットと呼んで区別する.

3.2 *WCS*の正当性

プロトコル *WCS*が同期時間 $12n$ の無待機時計合わせプロトコルであることを示す.

同期時間を, プロセッサが居眠りをしてから手続き $partial\_reset$ 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 2 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高々$6n$ 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.

第 93 行のソートの順序関係を表すため, 任意の 2 つの 2 項組 $(k_i, i), (k_j, j)$ に対し, $k_i < k_j$ または $k_i = k_j \wedge i < j$ならば, $(k_i, i) < (k_j, j)$ と定義する. また, 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, *3* 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, $(t_1, j_1) \preceq (t_2, j_2) \prec (t_3, j_3)$, $t+4n+1 \le t_1, t_3 \le t+5n$ 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.

(証明) P_{j_1}は, 時刻 $t_1 - n + j'$のステップで $P_{j'}$の共有変数を読み出し, P_{j_1}が世代を始めた時刻に対する $P_{j'}$が世代を始めた相対時刻 $P_{j_1}.key_{j'}$の値を決定し, 時刻 t_1で $sort_list$を行う. ここで, $P_{j_1}.key_{j'} = x$ とすると, $P_{j'} \longrightarrow^{j_1}_{t_1} P_{j_1}$ より $(x, j') > (0, j_1)$ である. 一方 P_{j_3}は, 時刻 $t_3 - n + j'$のステップで $P_{j'}$の共有変数を読み出して $P_{j_3}.key_{j'}$の値を決定し, 時刻 t_3で $sort\_list$ を行う. このとき, $t+4n+1 \le t_1 \le t_3 \le t+5n$ より $t_3 - 2n + j' < t_1 - n + j' \le t_3 - n + j'$が成り立つ. P_{j_3}は区間 $[t_3 - 2n + j', t_3 - n + j']$ では正常に動作し続けるので, $P_{j'}$がこの区間で居眠りをしているならば, 時刻 $t_3 - n + j'$のステップで P_{j_3}はそれを検知する. 時刻 $t_3 - n + j'$に P_{j_3}が $P_{j'}$の居眠りを検知しないなら以下の (a) が成り立ち, $P_{j'}$の居眠りを検知するなら以下の (b) が成り立つ.

(a) $(P_{j_3}.key_{j'}, j') = (x+t_3-t_1, j') > (t_3-t_1, j_1) \ge (t_3-t_2, j_2) = (P_{j_3}.key_{j_2}, j_2)$ ($\because$ $P_{j_2} \longrightarrow^{j_3}_{t_3} P_{j_3}$ より)

(b) $(P_{j_3}.key_{j'}, j') = (-1, j') < (0, j_3) = (P_{j_3}.key_{j_3}, j_3)$

(a),(b) はそれぞれ $P_{j'} \longrightarrow^{j_3}_{t_3} P_{j_2}$, $P_{j_3} \longrightarrow^{j_3}_{t_3} P_{j'}$ を意味する. ■

$steps(P_i, t', t)$ 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) $n \leq P_i.Work\_count(t-1) \leq 5n$ または $7n \leq P_i.Work\_count(t-1)$ 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, $work(P_i, t+6n) \geq 6n$ 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. $reader_s, dest_s (t+5n+2 \leq s \leq t+5n+\bar{\alpha})$ 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み出しをするならば $reader_{s-1} = reader_s$と定義し, 異なる共有変数の読み出しをするならば $reader_{s-1} = dest_s$ と定義する. ここで, $dest_s$ $(t + 5n+1 \leq s < t+5n+\bar{\alpha})$ のリスト $dlist = dest_{t+5n+1}, dest_{t+5n+2}, \cdots, dest_{t+5n+\bar{\alpha}}$ 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 $x(1 \leq x \leq h)$ 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 $x(1 \leq x \leq h)$ 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 $x(1 \leq x \leq h)$ 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, $work(P_i, t+6n) \geq 6n$ 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. 従って, $dlist^h$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から成り, $dlist^h$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 $P_{i_{x+1}}.Work\_count(s'_x - 1) \geq 5n$ が成り立ち, $(sort\_time^{x+1}, i_{x+1}) \preceq (sort\_time^h, i_h)$ が成り立つので, $2n+1 \leq P_{i_{x+1}}.Work\_count(t+3n) \leq 3n$ 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付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, $P_{i_x}.Work\_count(t+5n) \geq n$

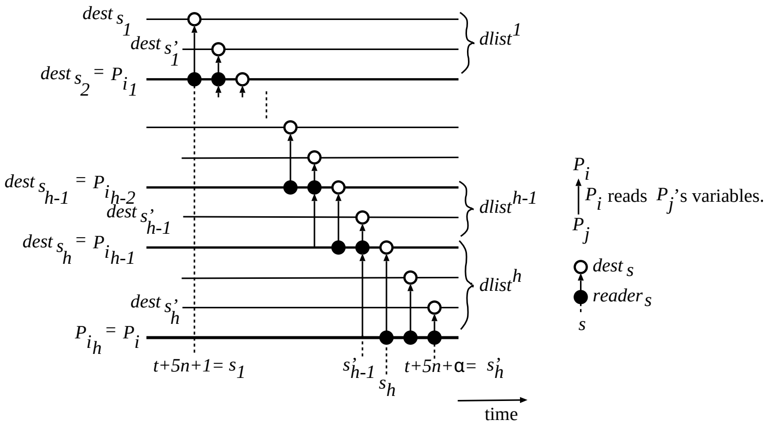

図 4: プロセッサ, 時刻の定義

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 $sort\_time^x \geq t+4n+1$ 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.

最後に (a) が成り立つことを示す. $(sort\_time^x, i_x) \preceq (sort\_time^h, i_h)$ 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から成り, $dlist^x$の要素はすべて異なる. 従って, (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 $(t+5n+1 \leq s \leq t+5n+\bar{\alpha})$ 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が表れない. 従って, $\bar{\alpha} < n$ 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 $x (1 \leq x \leq h)$ 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 $x, y (1 \leq x \leq h, 1 \leq y \leq h)$ 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 $x, y (1 \leq x < y \leq h)$ 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. 従って, $P_{j_1} = P_{i_x}$, $P_{j_2} = P_{i_{y-1}}$, $P_{j_3} = P_{i_y}$, $t_1 = sort\_time^x$, $t_2 = sort\_time^{y-1}$, $t_3 = sort_time^y$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. 従って, $dlist^x$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, 2 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．プロセッサ 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．

(証明) 時刻 t_i, t_jに関して場合分けする．

(1) $\max(t_i, t_j) \geq t - 2n$ のとき

一般性を失うことなく $(t_i, i) \prec (t_j, j)$ と仮定する．このとき，$t - 3n \leq t_j - n + 1 < t_j \leq t$ が成り立つ．P_i, P_jはともに区間 $[t_j - n + 1, t_j]$ で正常に動作し続け，この区間のある時刻 t'で P_jは $key[i]$ を設定するステップを行う．このとき，世代番号 $P_j.Gen(t)$ の時計調整で時刻 t 以前にプロセッサ P_jが P_iの居眠りを検知していないならば (第 57 行の条件式が成立)，$(t_i, i) \prec (t_j, j)$ より $((P_i.Work\_count(t' - 1) - P_j.Work\_count(t' - 1)), i) > (0, j)$ なので，$P_j \longrightarrow^{i}_{t_i} P_i$ が成り立つ．以下では，世代番号 $P_j.Gen(t)$ の時計調整で時刻 t 以前にプロセッサ P_jが P_iの居眠りを検知していないことを，背理法により示す．

$work(P_i, t) > 4n$ より，P_iは区間 $[t - 4n + 1, t]$ で居眠りをしていない．また，$2n \leq P_j.Work\_count(t - 3n) \leq 4n$ かつ $steps(P_j, t - 4n + 1, t - 3n + 1) = n$ なので，事実 2より区間 $[t - 4n + 1, t - 3n]$ のある時刻 t''で P_jは P_iの共有変数を読み出す．よって，P_j が世代番号 $P_j.Gen(t)$ の時計調整を始めてから P_j が t''より後で始めて P_iの居眠りに検知することはない． すなわち，時刻 t 以前に P_iの居眠りを検知するならば，遅くとも区間 $[t - 4n + 1, t - 3n]$ で必ず検知する．また，$P_i.Work\_count(t) \geq 5n$ かつ $steps(P_i, t - 2n + 1, t + 1) = 2n$ なので，事実 2より区間 $[t - 2n + 1, t]$ に P_iが P_jの共有変数を読み出すステップが必ず存在する．従って，区間 $[t'', t]$ で P_i が自身の居眠りを検知する．よって，区間 $[t'', t]$ のある時刻で P_iは $partial_reset$を行う．区間 $[t'', t]$ の長さは $4n - 1$ 以下なので，$P_i.Work\_count(t) < 4n$ となり矛盾する．

(2) $\max(t_i, t_j) < t - 2n$ のとき

一般性を失うことなく $i > j$と仮定する．以下では，$P_j \longrightarrow^{i}_{t_i} P_i$，$P_i \longrightarrow^{j}_{t_j} P_j$ がともに成立しないと仮定して矛盾を導く．プロセッサ P_iが t_i以前に最後に P_jの共有変数を読み出すステップを行った時刻を t'_iとし，P_jが t_j以前に最後に P_iの共有変数を読み出すステップを行った時刻を t'_jとする． $P_i \longrightarrow^{i}_{t_i} P_j$かつ $P_j \longrightarrow^{j}_{t_j} P_i$ が成り立つので，$P_i.Work\_count(t'_i - 1) \geq P_j.Work\_count(t'_i - 1)$ かつ $P_j.Work\_count(t'_j - 1) > P_i.Work\_count(t'_j - 1)$ であり，

$$steps(P_i, t'_i, t'_j) < steps(P_j, t'_i, t'_j) \tag{5.1}$$

が成り立つ．

ここで，$steps(P_i, t_i, t+1) > 2n$ と $P_i.Work\_count(t_i) = 5n$ が成り立つことより，$P_i.Work\_count(t) > 7n$ である．よって，事実 2より区間 $[t - n + 1, t]$ のある時刻 t''_iで P_iは P_jの共有変数を読み出す．同様にして，区間 $[t - n + 1, t]$ のある時刻 t''_jで P_jは P_iの共有変数を読み出す．P_i, P_jはそれぞれ区間 $[t_i, t], [t_j, t]$ で $partial_reset$を行わないので，

$$\begin{aligned} steps(P_i, t'_i, t''_i) &\geq steps(P_j, t'_i, t''_i) \\ steps(P_j, t'_j, t''_j) &\geq steps(P_i, t'_j, t''_j) \end{aligned}$$

が成り立つ．ここで，$work(P_i, t) \geq 4n, work(P_j, t) \geq 4n$ なので $steps(P_i, t''_i, t''_j) = steps(P_j, t''_i, t''_j)$ である．よって，

$$\begin{aligned} & steps(P_i, t'_i, t'_j) \\ = \ & steps(P_i, t'_i, t''_i) + steps(P_i, t''_i, t''_j) \\ & - steps(P_i, t'_j, t''_j) \\ \geq \ & steps(P_j, t'_i, t''_i) + steps(P_j, t''_i, t''_j) \\ & - steps(P_j, t'_j, t''_j) \\ = \ & steps(P_j, t'_i, t'_j) \end{aligned}$$

となり，式 (5.1) に矛盾する． ■

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 $\mathit{partial\_reset}$ 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．

(証明)　区間 $[t_{i,j}, t]$ で P_i が P_j の時計を棄却すると仮定し，矛盾を導く．すなわち，区間 $[t_{i,j}+1, t]$ のある時刻 $t_{i,m}$ で，調整ピリオドの P_i が P_m の共有変数を読み出したステップで，$P_i.Clock(t_{i,m}-1) \neq P_m.Clock(t_{i,m}-1)$，$\{P_j\} \in P_i.Sync(t_{i,m}-1) \subset P_m.Async(t_{i,m}-1)$ が成り立ち，P_i は P_m と局所時計を合わせた，または局所時計の一致確認をしたと仮定する．P_m が P_j を $P_m.Async$ に挿入した時刻を $t_{m,j}$ とし，$t_{m,j}$ より前で最後に P_m が P_j の共有変数を読み出した時刻を $t'_{m,j}$ とする．このとき，事実 2 より $steps(P_j, t'_{m,j}, t_{m,j}) < steps(P_m, t'_{m,j}, t_{m,j}) \leq 2n$ が成り立つ．ここで，$t_{m,j} \geq t-2n$ ならば，$steps(P_j, t'_{m,j}, t) = steps(P_j, t'_{m,j}, t_{m,j}) + steps(P_j, t_{m,j}, t) < 4n$ かつ P_j は時刻 $t'_{m,j}$ 以降に居眠りをしている．これは $work(P_j, t) > 4n$ に反する．また，$t_{m,j} < t-2n$ ならば，事実 2 より区間 $[t_{m,j}, t-1]$ に P_j が P_m の共有変数を読み出すステップが存在する．よって，$t'_{m,j}$ 以降に P_j は必ず $\mathit{partial\_reset}$ を行うので，仮定に反する．■

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，$work(P_i, t) > 4n, work(P_j, t) > 4n$ が成り立つならば，$P_i.Clock(t) = P_j.Clock(t)$ 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，$\mathit{sort\_list}$ を行っている．P_i, P_j が $\mathit{sort\_list}$ 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 $\mathit{partial\_reset}$ 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 $\mathit{partial\_reset}$ をすることはないので，それぞれステップ毎に局所時計の値を 1 ずつ増やす．従って，

$$steps(P_i, t', t) = steps(P_j, t', t) \qquad (7.1)$$

であることを示せばよい．ここで，$t' \geq t-4n+1$ 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 (7.1) 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 の局所変数を読み出しをするステップを行っており，また $t''(\leq t-n)$ より後で P_j は P_i の局所変数を読み出しを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 $\mathit{partial\_reset}$ 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．従って，$steps(P_i, t', t'') = steps(P_j, t', t'')$ である．さらに，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．従って，式 (7.1) は成り立つ．■

次の補題で，居眠りをしてから $\mathit{partial\_reset}$ 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 $\mathit{partial\_reset}$ を実行したならば，$work(P_i, t) < 6n$ が成り立つ．

(証明)　P_i が時刻 t で P_m の共有変数を読み出すとし，t より前で最後に P_i が P_j の共有変数を読み出すステップを行った時刻を t' とする．時刻 t の $\mathit{partial\_reset}$ 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．以下では，時刻 t の $\mathit{partial\_reset}$ が居眠りリセットである場合，すなわち (a) 第 42 行，(b) 第 43 行，(c) 第 74–75 行の条件式のうちいずれかが成り立つ場合を考える．

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

プロセッサ P_i は区間 $[t'+1, t-1]$ で居眠りをしている．$P_i.Work\_count \geq n$，事実 2 より $steps(P_i, t', t) \leq 2n$ であり，$work(P_i, t) < 2n$ が成り立つ．

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

P_j が P_i の居眠りを検知し $P_j.Invalid_i = P_i.Gen(t-1)$ にセットした時刻を t_j とする．t_j より前で最後に P_i が P_j の共有変数を読み出した時刻を t'_j とすると，

区間$[t'_j, t_j-1]$のある時刻でP_iは居眠りをしているので, 事実 2より $steps(P_i, t'_j, t_j) < steps(P_j, t'_j, t_j) \leq 3n-1$が成り立つ. さらに, 区間$[t_j+1, t-1]$には$P_i$が$P_j$の共有変数を読み出すステップが存在しないので, $P_i.Work\_count(t-1) \geq n$, 事実2より $steps(P_i, t_j+1, t) \leq 2n$ が成り立つ. 従って, $steps(P_i, t'_j, t) < 5n$ であり, 区間 $[t'_j, t_j-1]$ のある時刻で P_iは居眠りをしているので $work(P_i, t) < 5n$ が成り立つ.

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

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 $5n \leq P_i.Work\_count(t_m-1) < P_i.Work\_count(t-1)$ 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, $work(P_i, t) \geq 6n$ 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, t 以前で P_iが最後に $partial_reset$を行った時刻を t^0とする. すなわち, $Work\_count(t^0) = 0$ とする. t 以前に $partial\_reset$ を行っていないならば, $t^0 = 0$ とする. $work(P_i, t) \geq 6n$ 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. 従って, $work(P_m, t_m-1) \geq t_m - t^0 - n > 4n$, $work(P_j, t-1) > work(P_j, t_m-1) > 4n$ 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 $Work\_count$ の値を基に $sort_list$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. 従って, 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. 従って, $steps(P_j, t'_m, t) = steps(P_m, t'_m, t_m) + steps(P_i, t_m, t)$ 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, $work(P_i, t-1) \geq 12n$ 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, $work(P_i, t) \geq 12n$ 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, $P_i.Work\_count(t-1) \geq 6n$ 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$t \geq 1$, 任意のプロセッサP_i, P_jに対し, $work(P_i, t) \geq 12n, work(P_j, t) \geq 12n$ 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$12n$の無待機時計合わせプロトコルである. ■

4 同期時間の下界

本節では, フェーズ内システムにおける無待機時計合わせプロトコルの同期時間の下界が$\Omega(n)$であることを示す.

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

(証明) 同期時間 $k \le n-2$ の時計合わせプロトコル A が存在すると仮定し, 矛盾を導く. 次の 3 つの条件を満たす 2 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 $work(P,t) \ge k$ が成り立つ.

このとき, 両実行 E, E'において, 一致性より, すべてのプロセッサ P_iに対し $P_i.Clock(t)$ が一致する. この値を $clock(t)$ とする.

(条件 2) 実行Eに対し, $\pi_{t+1} = \pi_{t+2} = \cdots = \pi_{t+n-1} = \{P_0\}$ 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は区間 $[t+1, t+n-2]$ で $n-2$ ステップの動作をするので, その間に読み出しをしていない共有変数集合 $V_i(i \ne 0)$ が存在する.

(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 (12,1) $P_i.Clock_{E'}(t+n-1) = clock(t)+n-1$ 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$[t+1, t+n-1]$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, (12,2)$P_0.Clock_{E'}(t+n-1) = P_0.Clock_E(t+n-2) = clock(t)+n-2$ 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, 式(12, 1), (12,2) に矛盾する. ■

以上より, 無待機時計合わせプロトコルの同期時間の下界は$\Omega(n)$ である. 従って, 前節で提案したプロトコル WCSの同期時間はオーダー的に最適である.

5 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わ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 $12n$ の無待機時計合わせプロトコル WCSを提案し, このプロトコルが同期時間に関してオーダー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わ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始めとする本学情報論理学講座の皆様方に深く感謝いたします . 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.

参考文献

[1] D. Dolev, J. Halpern and H. Strong: "On the possibility and impossibility of achieving clock synchronization," *Journal of Computer System Science*, 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.