マルチプロセッサ上での細粒度並列処理のための1対1同期機構

早川 潔[†] 本多弘樹[††]

マルチプロセッサ上での細粒度並列処理において，タスク間の先行制約保証のための制御を効率良く行うためには，同期コストの低いハードウェア同期機構が必要となる．本論文では，不必要な待ち時間がいっさい生じないハードウェア同期機構—1対1同期機構—とそれを活用するためのコンパイル手法を提案する．本方式は，不必要な待ち時間がいっさい生じないハードウェア同期機構としては，比較的動作が単純でコード生成がしやすく，またハードウェア量も少ないという特徴を持つ．さらに本論文では，実マルチプロセッサ上で1対1同期機構を用いた細粒度並列処理を実現し，1対1同期機構の有効性を検証する．

The One-on-One Hardware Synchronization Mechanism for Fine-grain Parallel Processing on the Multiprocessor

KIYOSHI HAYAKAWA[†] and HIROKI HONDA[††]

For the efficient execution of synchronization on fine-grain parallel processing, hardware synchronization mechanisms are needed. In this paper, we propose a new hardware mechanism — One-on-One hardware synchronization mechanism — which has no unnecessary wait time and propose the compiler method for use of this mechanism. The features are that the compiler with this method can easily produce synchronization codes and that the mechanism can be made with less amount of hardware. We also evaluate the performance of the One-on-One hardware synchronization mechanism implemented on the experimental MIMD machine.

1. はじめに

マルチプロセッサ上の細粒度並列処理[4)~6)]では，「先行/後続タスク[6)]間で，先行タスクの実行が終了しなければ後続タスクの実行が開始できないようにするための同期（タスク間同期）」が行われる．細粒度並列処理を効率良く行うには，タスク間同期の処理時間が細粒度タスクの実行時間と比べて十分に小さくなければならない．このため，タスク間同期をハードウェアで行うことにより同期処理時間の小さい同期機構—ハードウェア同期機構—が提案されている[2),8)~10)]．

ハードウェア同期機構としては，高速な同期を可能にすると同時に，動作が単純でコンパイラによる同期コードの生成が容易で，また少ないハードウェア量で実現できるものが望ましい．

ハードウェアによる実現が容易な同期モデルの1つとして，バリア同期モデルがあげられる．タスク間同期にバリア同期を用いる場合，先行制約関係にないタスク間に同期機構依存先行制約[10)]と呼ばれる余分な先行制約が付加されてしまう．この余分な先行制約が付加されたタスク間に，待ち時間が生じてしまう場合がある．本論文では，この待ち時間を不必要な待ち時間と呼ぶことにする．

不必要な待ち時間を少なくすることを目的として，強制参加&線状[13)]のバリア同期を用いる際にタスクスケジューリングを工夫する方法[11)]やバリア同期の概念を一部拡張し同期機構依存先行制約を減らす方法[2),8)~10)]が提案されている．

前者の方法では，タスクの見積り実行時間と実際の実行時間の差（タスクの実行時間変動）が大きいほど，実際の同期ポイントの実行時刻がずれて，不必要な待ち時間が生じてしまう可能性が高くなる．また，後者の方法でも，同期機構依存先行制約を完全になくすこ

† 山梨大学電子情報工学科
Computer Science Department, Yamanashi University
†† 電気通信大学大学院情報システム学研究科
Graduate School of Information Systems, University of Electro-Communications

とはできない．

一方，先行制約ごとに，先行タスク側ではその実行終了を知らせる操作（先行タスク終了通知），後続タスク側ではその先行タスクの実行終了を検査する操作（先行タスク終了検査）を行い，実行時には先行タスク直後の先行タスク終了通知の実行が開始しなければ，後続タスク直前の先行タスク終了検査の実行が終了しないように，制御する同期モデル（通知/検査同期モデル）が考えられる．このモデルでは，同期機構依存先行制約がなく，不必要な待ち時間は存在しないという利点がある．

この同期モデルを実現する方法として，各先行制約に1つずつフラグ変数を用意し，その変数に対してセットとテストを行う方法（Binary Semaphore に対する P/V 操作と類似した操作）が考えられる．しかし，この方法をハードウェアで実現するには，フラグ変数の数が多くハードウェアコストが大きい．

本論文で対象とする細粒度並列処理（後述）では，コンパイル時に各タスクの実行順番と実行するプロセッサを決定しているので，タスク間同期を行うべきプロセッサ（対）とその同期の順序もコンパイル時に決定することができる．このことを利用すると，先行制約すべてにフラグ変数を用意するのではなく，各プロセッサ対に通知/検査のための変数を用意すればよく，変数の数を減らすことができる．このことを利用して通知/検査同期モデルを実現するハードウェア同期機構として，一般化静的順序制御機構[10]が提案されている．一般化静的順序制御機構では，各プロセッサ間に2つの変数を用意し，その変数に対するカウントアップとテスト&デクリメント操作（General Semaphore に対する P/V 操作と類似した操作）を行う．この変数は，プロセッサ N 台に対して，$N(N-1)$ 個必要となる．また，プロセッサのレイアウトによっては，同期機構のためのプロセッサ間配線数が $O(N^2)$ になる．

そこで，本論文では，同期機構のためのプロセッサ間配線数が比較的少なく，通知/検査同期モデルを実現するハードウェア同期機構/コンパイル方式として，1対1同期機構とそれを活用するためのコンパイル方式（1対1同期方式☆）を提案する．また，1対1同期機構を実マルチプロセッサにインプリメントし，実機上で性能評価を行った結果を述べる．

☆ ここでいう同期機構とは，同期処理回路そのものを意味し，同期方式とは，同期機構を使用してタスク間同期を行う方法を意味する．

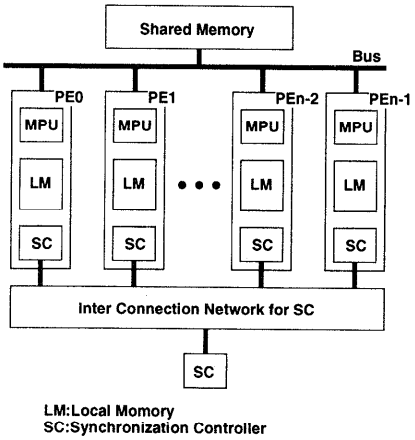


図1 対象とする並列計算機モデル
Fig. 1 The target multiprocessor.

2. 前提とする並列処理方式

本論文で前提とする並列処理方式[6]は，逐次プログラムの基本ブロックを細粒度タスクに分割し，そのタスク間の並列性を利用して並列処理するものである．本方式では，1) 基本ブロックの細粒度タスクへの分割，2) タスク間の並列性の検出，3) タスク集合のプロセッサへのスケジューリング（各プロセッサで実行すべきタスクとその順序の決定），4) データ通信と同期が必要な位置の決定とそのコードの生成，5) 各タスクコードの生成，6) 各プロセッサ用オブジェクトコードの生成は，並列化コンパイラによって行われる．

本並列処理方式の前提とする並列計算機は，プロセッサ台数が十数から数十の共有メモリ型マルチプロセッサである（**図1**）．各プロセッサで実行されるプログラムは，Local Memory（LM）に格納され，MPU によって逐次実行される．1対1同期機構を実現するために，Synchronization Controller（SC）を各 PE 内に実装する．各 PE 内の SC どうしは，相互結合網で相互に結合される．

3. 1対1同期機構

3.1 原理：タスク間同期の実現

1対1同期機構では，各プロセッサごとにそのプロセッサで先行タスクをいくつ実行（終了）したかを示す**先行タスクカウンタ**を用意し，そのカウンタに対するカウントアップとテスト操作（General Semaphore の P/V（特定の値と比較する）操作と考えることもできる）を行う．

先行タスク終了通知を行う同期コード（以後通知コードと呼ぶ）および先行タスク終了検査を行う同期

コード（以後，検査コードと呼ぶ）は，次のような動作を行う．

通知：通知コードを実行しているプロセッサの先行タスクカウンタをインクリメント．

検査（PE_i，N_i）：対応する通知コードを実行するプロセッサ PE_i（i はプロセッサ番号）の先行タスクカウンタの値（Nc_i）(プログラム実行時の観測値）と，その通知コードのプロセッサ内での実行順番（N_i）(コンパイル時に求まる値）から，通知コード実行終了条件（$Nc_i \geq N_i$）が成立しているか否かを検査．条件が成立している場合，検査コードの実行を終了．成立していない場合，成立するまで検査コードの実行終了を延期．

N_i の値はコンパイル時に決定される定数である．一方，Nc_i の値は検査コード実行時刻までに PE_i で実行された通知コードの個数であり，観測される Nc_i の値は検査コードの実行時刻によって変動する．$Nc_i \geq N_i$ であれば，対応する通知コードの終了が保証されている．

3.2 先行タスクカウンタの設置場所

上記のタスク間同期を行うために，原理的には，先行タスクカウンタを各プロセッサに対して1つ用意すればよい．そのカウンタと比較回路の最も簡単な設置場所は，各プロセッサ内にそのプロセッサに対する先行タスクカウンタと比較回路を設置することである．検査コードを実行するプロセッサは，対応する通知コードを実行するプロセッサの比較回路に比較値を送出をすればよい．

しかし，各プロセッサをバスで結合した場合，このようなカウンタと比較回路の配置では，バスに送出される比較値のデータバス配線量が増大しハードウェアコストが大きくなる．

そこで，カウンタ数はプロセッサ N 台に対して $N(N-1)$ 個必要となるが，各プロセッサ内に他のすべてのプロセッサの先行タスクカウンタとそれに付随する比較回路を用意し，バス上にはカウントアップの信号線を各プロセッサに1本用意したほうが，バスの配線数を少なくすることができる．このように構成した1対1同期機構の同期検出回路（プロセッサ4台時）を**図2**に示す．

3.3 1対1同期機構の構成

同期検出回路は，カウントアップ回路（count up），先行タスクカウンタ回路（count），比較器（comp）およびAND回路で構成される．data信号，up信号，end信号およびselect信号は，通知/検査コード実行時にプロセッサ内にあるMPUによって送出される．

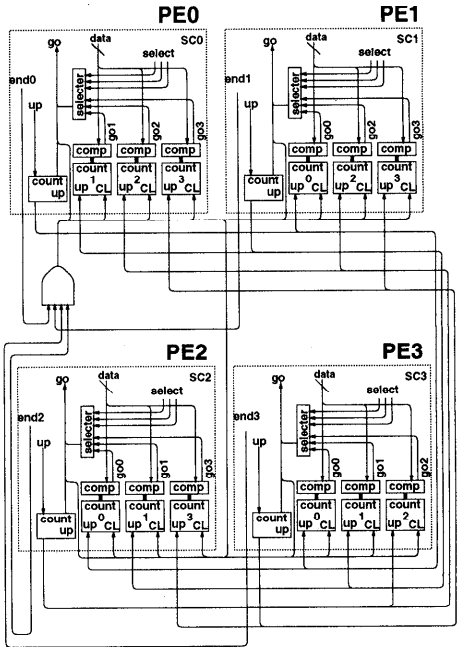


図2 1対1同期機構の構成（PE 4台）
Fig. 2 The One-on-One synchronization mechanism (4 PEs).

カウントアップ回路とそれに対応する先行タスクカウンタ回路（群）は，1本の信号線で結合される．

3.4 1対1同期機構の動作

通知コードを実行するMPUが，図2のup信号を出力することにより，先行タスクカウンタをインクリメントさせる．

検査コードを実行するMPUは，その検査コードに対応する通知コードの N_i をdata信号線を介して比較回路に送出し，比較回路からgo信号が返ってくるまで動作を停止する．go信号が返ってきたら，検査コードを終了し，後続の命令実行に移行する．比較回路は，MPUから送られた値（N_i）と先行タスクカウンタの値（Nc_i）を比較し，通知コード実行終了条件が成立している場合，go信号をMPUに送出する．成立していない場合，go信号は送出せず比較の状態を続ける．

比較回路の配線数コストを小さくするために，これらの比較は，まずSC内の全先行タスクカウンタに対して行い，必要な比較結果をselect回路で選択する．

なお，通知コードの総数が先行タスクカウンタの上限値を超えてしまう場合には，通知コードの総数がその上限を超える直前に，end信号を送出し，バリア同期を行い，カウンタの値を0にする．

4. 並列化コンパイラ

1対1同期機構用の同期コードを含んだ並列オブジェクトプログラムは，以下の手順で並列化コンパイラによって生成される．

(1) ソースプログラムをステートメントレベルの細粒度タスクに分割する．

(2) 分割したタスク間のデータ依存解析を行いタスクグラフを生成する．

(3) タスクグラフを基にCP/MISF法[3]を用いたタスクスケジューリングを行い，各プロセッサで実行すべきタスクとその順序を決定する．

(4) データ依存関係およびスケジューリング結果から異なるプロセッサに割り当てられたタスク間の先行制約を決定する．文献7)の手法を使用し，冗長な先行制約を削除する．

(5) (4)で求めた先行制約関係にある先行/後続タスクを求め，先行タスクの直後に，通知コードを配置する．また，各通知コードが実行されるプロセッサとそのプロセッサ内での実行順番を求める．

(6) (5)で求めた後続タスクの直前に，検査コードを配置する．その際，検査コードに必要なselect信号値および比較値（N_i）は，(5)で求められた先行タスクに対応する通知コードのプロセッサ番号およびそのプロセッサ内での実行順番から決定される．

(7) 各タスク（データ通信コードと同期コードを含む）に対応するオブジェクトコードを生成する．

5. 評価システム：OPAS

1対1同期機構の評価を行うために用いたマルチプロセッサシステムOPASは，各種ハードウェア同期機構の研究用テストベッドとして開発した共有メモリ型MIMDマルチプロセッサシステムである（**図3**）．4台のProcessing Element（PE）とShared Memory（S-RAM: 1 MByte）は，1本のVMEバスで結合される．各PEは，MPU（MC68HC000（8 MHz））とローカルメモリ（256 KByte）で構成される．Control Processor（CP）は，並列オブジェクトプログラムやデータの授受に使用される．Synchronization Controller（SC）は，1対1同期機構用のSC_i $(i = 0 \ldots 3)$とSBM同期機構[9]用のSC（SCS）の2種類で構成した．SCSは，PE Wait・Go信号線で各PEと結合され，同期成立の検出・PE Go信号の送出を行う．SC_iは，互いにcount up信号線で結合され，count up信号の送出，先行タスクカウンタ値の比較を行う．

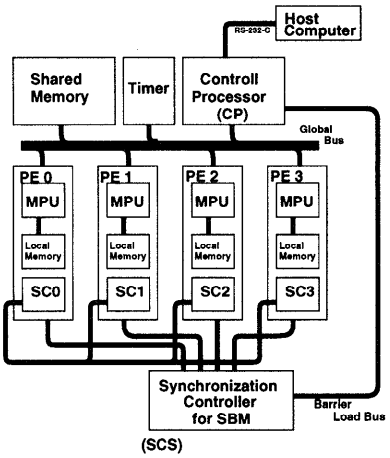


図3 OPASシステムのハードウェア構成
Fig. 3 The OPAS's architecture.

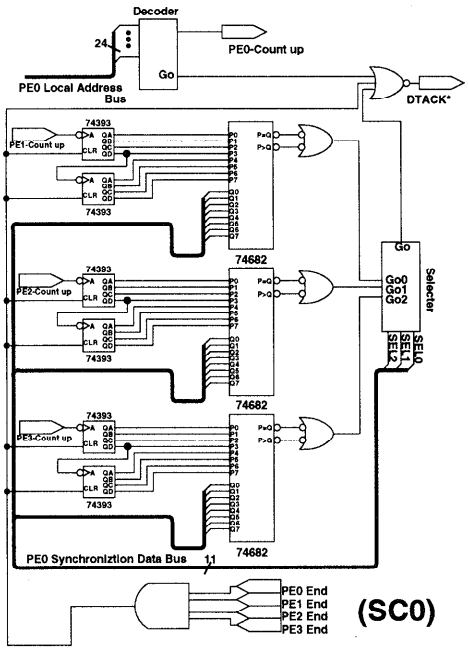


図4 1対1同期機構の同期検出回路図（SC0）
Fig. 4 The synchronization circuit of the One-on-One synchronization mechanism (SC0).

1対1同期機構の同期検出回路は，**図4**のように構成した．先行タスクカウンタは，非同期カウンタの74393，比較器は，8-Bit Magnitude Comparatorの74682で構成した．Decoderは，GALと74138で構成され，count up信号の送出を行う．

通知コードは，特定番地への読み込み命令で実現した．その動作を以下に示す．

(1) MPU が FFF002 番地にアクセス.
(2) FFF002 番地のアクセスを検出した Decoder 回路は，conut up 信号と go 信号を送出.
(3) go 信号を MPU の DTACK*信号とし，SC_i が DTACK*信号をアサート.

検査コードは，特定番地への書き込み命令で実現した．動作を以下に示す.

(1) MPU が FFF004 番地に先行タスクカウンタの select 信号値（SEL0～SEL2）と N_i を書き込む.
(2) select 信号値および N_i を Synchronization Data Bus に送出.
(3) Synchronization Data Bus のデータの値と比較回路の判定信号（$P=Q$, $P>Q$）の値が定まったら，SEL 信号で指定した go$i$ 信号を go 信号にする.
(4) go 信号を MPU の DTACK*信号とし，SC_i が DTACK*信号のアサート.

6. OPAS 上での性能評価

性能評価は，各テストプログラムに対し以下の 3 種類の同期方式を用いて並列処理を行い，実行時間を比較した.

- 1 対 1 同期機構を用いた 1 対 1 同期方式
- SBM 同期機構を用いた One-PE 同期方式
- 共有メモリ上のフラグ変数を用いたフラグチェック同期方式

One-PE 同期方式[2]とは，バリアキューによるバリア実行の順序づけ（ブロッキング）から生じる先行制約を利用し，SBM 同期機構上でタスク間同期を行うもので，高速な同期動作が可能であるが，タスクの実行時間変動が大きい場合，不必要な待ち時間が長くなる.

One-PE 同期方式と 1 対 1 同期方式の性能を比較することにより，同期モデルの差（同期機構依存先行制約から生じる不必要な待ち時間の差）や実装の差（通知/検査コードの数）がどのように影響しているか考察する.

フラグチェック同期方式とは，前述した共有メモリに用意されたフラグ変数への set と busy wait によって，通知/検査同期モデルを実現する同期方式である.

テストプログラムは，C 言語で記述した．プログラム中の各基本ブロックに対して並列処理を行う．C のソースプログラムを著者らが開発した C 並列化コンパイラによりコンパイルし，生成された並列オブジェクトプログラムを OPAS システムで並列実行した.

6.1 性能評価 1

画像処理の一種であるディザ処理を行うプログラムの一部を並列実行した．このプログラムでは，共有メモリアクセスディレイと乗除算命令の実行時間変動によるタスクの実行時間変動がある．プロセッサ台数と実行時間の関係を**図 5** に示す.

1 対 1 同期方式の実行時間は，One-PE 同期方式のそれより短いという結果を得た．両同期方式では，タスク割当てが同じで，同期コード数の差もごくわずかである（**表 1**）ことから，この結果は，Onc-PE 同期方式で生じた同期機構依存先行制約による不必要な待ち時間が原因であると考えられる.

フラグチェック同期方式と 1 対 1 同期方式では，同期コードの数，プログラムのタスク割当ておよび同期コードの配置は同じである．また，1 対 1 同期方式における同期コードの最小実行時間は 3.0 [μsec] であるのに対して，フラグチェック同期方式では 4.0 [μsec] と大差ない．このことから，フラグチェック同期方式に

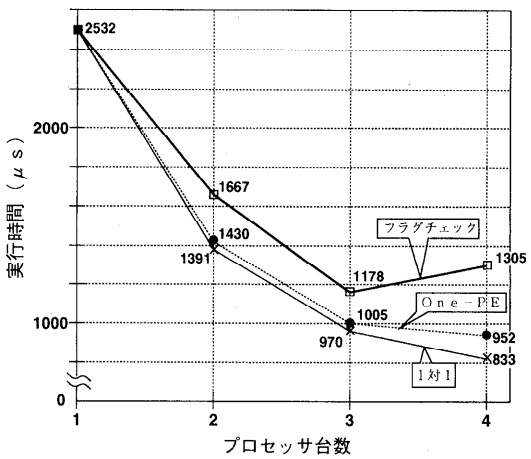


図 5 評価プログラム 1 における実行時間とプロセッサ台数の関係
Fig. 5 Processing times vs. number of PEs of prog.1.

表 1 1 対 1 同期方式と One-PE 同期方式の同期コード数の差
Table 1 The difference between the number of One-on-One's sync. codes and One-PE's sync. codes.

プログラム No.		1	2		
プロセッサ台数		4	2	3	4
全プロセッサの同期コード総数	1 対 1	106	26	52	76
	One-PE	102	26	43	53
同期コードの差（全プロセッサ）		4	0	9	23
同期コードの差（各プロセッサ）		0～2	0	2～4	5～6

（プログラム 1 のプロセッサ台数は，プロセッサ台数 2，3，4 台時で，全プロセッサが実行する同期コード数の差が一番大きいときの台数）

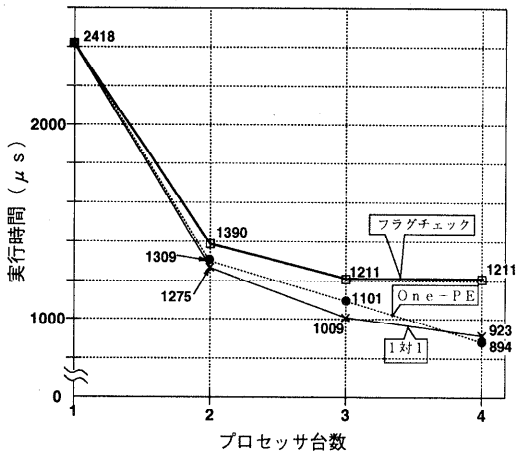


図6 評価プログラム2における実行時間とプロセッサ台数の関係
Fig. 6 Processing times vs. number of PEs of prog.2.

おけるメモリコンテンションがフラグチェック同期方式と1対1同期方式の実行時間の差の原因と考えられる.

6.2 性能評価2

差分法による流体のシミュレーションを行うプログラムの一部[6]を並列実行した. このプログラムでも, 前例題と同様のタスクの実行時間変動がある. プロセッサ台数と実行時間の関係を図6に示す. プロセッサ2・3台では, 1対1同期方式の実行時間がOne-PE同期方式のそれよりそれぞれ3%・8%短く, プロセッサ4台では3%長いという結果を得た.

プロセッサ4台で1対1同期方式がOne-PE同期方式より実行時間が長くなった原因の1つとして, 1対1同期方式の同期コード数がOne-PE同期方式のそれより特に多くなってしまったことが考えられる（表1参照）.

6.3 性能評価3

タスクの実行時間変動が大きい場合の並列実行時間への影響を調べるために, 性能評価2のタスクの実行時間を意図的に変動させたプログラムをプロセッサ4台で並列実行した.

評価プログラム3の作成方法を以下に示す.

(1) タスクの実行変動を考慮に入れず, 性能評価2で使用したプログラムの並列オブジェクトプログラムを作成する.

(2) 並列オブジェクトプログラム中のタスクからランダムにタスクを選び, その先頭にNOP命令をループさせるコードを付け加える. ループの回数もランダムに選ぶ. 1ループで20クロック遅れる.

表2 各評価プログラムの変動タスク数と平均NOP数
Table 2 The number of fluctuation tasks and the average number of NOPs.

プログラムNo.	変動タスク数	平均NOP数
2	6	17.8
3	12	20.2
4	7	36.1
5	8	41.6
6	11	33.3
7	12	31.8
8	18	27.7
9	20	27.4
10	22	26.3
11	24	29.0

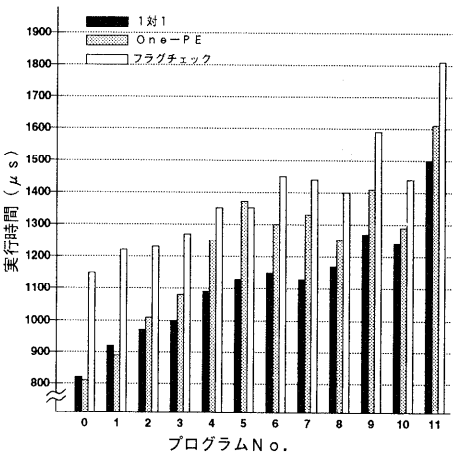


図7 タスクの実行時間を変動させたプログラムに対する各同期方式の性能（プロセッサ4台）
Fig. 7 Processing time for the programs including fluctuation tasks.

以上の方法を用いて, 10種類の評価プログラムを作成した.

各評価プログラムの実行時間を変動させたタスク数とそれらのタスクに付け加えたNOP数の平均を表2に示す. (変動タスク数 × 平均NOP数) が小さい順にプログラムNo.の番号をつけた. プログラムNo.0では, 評価プログラム2の中にある変数乗除算を定数乗除算とし, 乗除算によるタスク実行時間の変動がなく, より見積り実行に近い結果が得られるようにしている. プログラムNo.1は, 性能評価2で用いたプログラムそのものである. 各プログラムに対する各同期方式での実行時間を図7に示す.

この結果から, タスクの実行時間が見積り実行時間からずれるほど, 1対1同期方式とOne-PE同期方式との性能差が大きくなることが分かる. No.0, No.1のプログラムでは, 1対1同期方式の同期コード数がOne-PE同期方式より多いことによる実行時間の延び

と，One-PE 同期方式の同期機構依存先行制約で生じる不必要な待ち時間による実行時間の延びがほぼ同じであるため，ほぼ同じ実行時間になったと考えられる．しかし，No.2～11 のプログラムでは，One-PE 同期方式の同期機構依存先行制約で生じる不必要な待ち時間による実行時間の延びが，1 対 1 同期方式の同期コード数が One-PE 同期方式より多いことによる実行時間の延びより大きくなったため，1 対 1 同期方式の実行時間が One-PE 同期方式のそれより短くなったと考えられる．

7. 関連研究との比較

高速な同期を可能とするハードウェア同期機構として，SBM 同期機構[9]，Fuzzy Barrier[1]，Elastic Barrier[8]，重複可能なバリア型同期機構[12]，一般化静的順序制御機構[10]があげられる．本章では，これらの同期機構との比較を行う．

［同期コード数］

OPAS と同様に一般的な MPU を用いた場合，タスク間同期は，同期コードをプログラム中に配置することにより実現される．この場合，同期コードの数が多いと並列処理の効率を低減させる一因となる．

Fuzzy Barrier・重複可能なバリア型同期機構では，あるタスク間の先行制約を満たすために，そのタスクを実行しないプロセッサにも入口・出口コードが必要であるが，1 対 1 同期機構では必要ない．SBM 同期機構とそれを利用した One-PE 同期方式では，ある先行制約を保証するための同期コードが他の先行制約も保証する場合があるので，1 対 1 同期機構より同期コード数を少なくできる．

後続タスクを実行するプロセッサ以外の複数のプロセッサに複数の先行タスクが割り当てられた場合，本論文でインプリメントした 1 対 1 同期機構では，先行タスクを実行するプロセッサの数だけ検査コード数が必要となり，一般化静的順序制御機構より同期コード数が多くなってしまう．これに対しては，後述するとおり，同期検出回路を工夫することにより，これを減らすことができる．

［同期条件が成立していることを検出可能となる時点］

同期コードを MPU によるデータ書き込み動作によって実現する場合，1 対 1 同期機構の検査コードで同期検出可能となる時点は，バリア型同期機構[12]の同期コードより遅れる．これは，バリア型同期機構の同期コードでは，アドレスを送出した時点で同期検出が可能となるのに対して，1 対 1 同期機構の検査コードでは，アドレス送出後にデータ書き込みをした時点となるためである．OPAS では，同期命令のクロック数は，23 クロック以上に対しこの遅れは 0～3 クロックとなる．なお，この遅れを 5 ステージパイプライン処理を行う MPU で試算すると同期命令クロック数 9 クロック以上に対して 0～3 クロックの遅れとなる．

［ハードウェア量］

1 対 1 同期機構と一般化静的順序制御機構のハードウェア量を比較した場合，以下の点が異なる．

- 各 PE ボードが VME バスなどのバックプレーンボードに 1 枚ずつ接続されており，かつ各 PE ボード内にトークン（または先行タスク）カウンタが搭載されていると仮定すると，プロセッサ間同期結合網の配線数は，一般化静的順序制御機構ではプロセッサ N 台に対して $O(N^2)$ 本，1 対 1 同期機構では $O(N)$ 本となる．
- 3.2 節で述べたように，1 対 1 同期機構では，先行タスクカウンタを複製し，他のプロセッサに用意しているため，プロセッサ N 台に対して $N(N-1)$ 個の先行タスクカウンタが必要となる．しかし，全プロセッサと先行タスクカウンタを 1 つの VSLI 内に搭載した場合，先行タスクカウンタを複製して用意する必要がなく，各プロセッサに 1 つ用意すればよい．この場合，先行タスクカウンタの数は，プロセッサ N 台に対して N 個となる．一方，一般化静的順序制御機構では，どのようにプロセッサをレイアウトしても，$N(N-1)$ 個のトークンカウンタが必要になる．
- 一般化静的順序制御機構のトークンカウンタ（UP/DOWN カウンタ）では，1 つのトークンカウンタの UP 信号と DOWN 信号との間にアービトレーション回路を設けるか，アップカウンタとダウンカウンタの 2 つのカウンタを用意しそれらのカウンタの値を比較する回路にしなければならない．一方，1 対 1 同期機構では，先行タスクカウンタは UP カウンタのみで構成（その代わり，ときどきカウンタリセットが必要になる）できる．

8. おわりに

1 対 1 同期機構を提案し，実マルチプロセッサ上で性能評価を行った．性能評価では，以下の結果を得た．

- タスクの実行時間変動が少ない場合，1 対 1 同期方式では One-PE 同期方式と同等またはそれ以上の性能が得られる．
- タスクの実行時間変動が大きいほど，One-PE 同期方式より 1 対 1 同期方式の性能が良くなる．

今後の課題として，以下のことがあげられる．

- 1 対 1 同期機構の同期コード数を減らすため，同期

検出回路を改良（図4のSynchronization Data Bus（D0～D15）をD0～D4, D5～D9, D10～D14とカウンタ数に分割し，各比較器の入力にはこの分割したデータバスの1つを入力させることにより，検査コード1命令で複数の比較動作が可能となり，後続タスクの同期コード数を減らすことができる）し，性能評価を行う．

- 1つの後続タスクに対し先行タスクが複数存在するか1つなのかによって，One-PE同期方式と1対1同期方式を使い分けるといったハイブリッドな方式を検討する．
- 他の様々な評価プログラムに対して性能評価を行う．

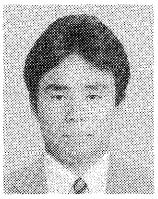
参考文献

1) Gupta, R.: The Fuzzy Barrier: A Mechanism for High Speed Synchronization of Processors, *Proc. 3rd Int'l. Conf. Architectural Support for Programming Languages and Operating System*, pp.54–63 (1989).
2) 早川 潔，増村 均，本多弘樹：SBM同期機構を用いたOne-PE同期方式，電子情報通信学会論文誌，Vol.J78-D-I, No.2, pp.73–81 (1995).
3) Kasahara, H. and Narita, S.: Practical Multiprocesssor Scheduling Algorithms for Efficient Parallel Processing, *IEEE Trans. Comput.*, Vol.C33, No.11, pp.1023–1029 (1984).
4) Kasahara, H., Honda, H., Kai, M., Seki, T. and Narita, S.: Parallel Processing for Simulation of Dynamical System, *Proc. IFAC 7th Conf.*, pp.527–533 (1985).
5) Kasahara, H., Honda, H. and Narita, S.: Parallel Processing of Near Fine Grain Tasaks Using Static Scheduling on OSCAR, *Proc. Supercomputing '90*, pp.856–864, ACM & IEEE (1990).
6) 本多弘樹，水野 聡，笠原博徳，成田成之助：Fortranプログラム基本ブロックの並列処理手法，電子情報通信学会論文誌，Vol.J73-D-I, No.9, pp.756–766 (1990).
7) Phillip, L.S.: Minimaization of Interprocessor Synchronization in Multiprocessors with Shared and Privete Memory, *Proc. Int'l. Conf. Parallel Processing*, Vol.III, pp.138–142 (1990).
8) 松本 尚：細粒度並列処理実行支援マルチプロセッサの検討，情報処理学会論文誌，Vol.31, No.12, pp.1840–1851 (1990).
9) O'Keefe, M.T. and Dietz, H.G.: Hardware Barrier Synchonization: Static Barrier MIMD (SBM), *Proc. Int'l. Conf. Parallel Processing*, Vol.I, pp.35–42 (1990).
10) 高木浩光，有田隆也，曽和将容：問題が持つ先行関係のみを保証する高速な静的実行順序制御機構，情報処理学会論文誌，Vol.32, No.12, pp.1583–1592 (1991).
11) 高木浩光，有田隆也，川口喜三男，曽和将容：バリア同期のためのタスクスケジューリングアルゴリズムとその性能評価，情報処理学会研究報告，94-ARC-105, pp.73–80 (1994).
12) 高木浩光，有田隆也，曽和将容：細粒度並列実行を支援する種々の静的順序制御方式の定量的評価，並列処理シンポジウムJSPP '91, pp.269–276 (1991).
13) 山家 陽，村上和彰：バリア同期モデル—Taxonomyと新モデルの提案，および，モデル間性能比較，並列処理シンポジウムJSPP '93論文集，pp.119–126 (1993).

(平成8年8月19日受付)
(平成9年5月 8 日採録)

早川 潔（学生会員）

昭和42年生．平成4年山梨大学工学部電子工学科卒業．平成6年山梨大学大学院工学研究科電子情報工学専攻博士前期課程修了．同年同大学院博士後期課程入学．現在に至る．ハードウェア同期機構等の並列計算機のアーキテクチャの研究に従事．電子情報通信学会，IEEE各会員．



本多 弘樹（正会員）

昭和36年生．昭和59年早稲田大学理工学部電気工学科卒業．平成3年早稲田大学大学院博士課程修了．昭和62早稲田大学情報科学教育センター助手．平成3年山梨大学工学部電子情報工学科専任講師．平成4年同助教授．平成9年電気通信大学大学院情報システム学研究科助教授．現在に至る．並列処理方式，並列化コンパイラ，マルチプロセッサアーキテクチャなどの研究に従事．工学博士．電子情報通信学会，IEEE，ACM各会員．