計算機アーキテクチャ　105−11
ハイパフォーマンス
コンピューティング　50−11
（1994．3．11）

1対1ハードウェア同期機構の構成法及び性能評価

早川 潔　増村 均　本多 弘樹

山梨大学電子情報工学科

細粒度並列処理における同期処理を効率良く行なうために、同期コストの低いハードウェア同期機構が必要となる。しかし、先行制約を保証するために同期を行なう場合、ハードウェア同期機構の種類によっては不必要な待ち時間が生じてしまう。本稿では、先行制約を満たすハードウェア同期機構として、不必要な待ち時間が一切生じない1対1ハードウェア同期機構の構成法について述べる。また、1対1ハードウェア同期機構を実マルチプロセッサ上にインプリメントし、実機上で他の同期方式と比較することにより1対1ハードウェア同期機構の有効性を検証する。

Organization and Performance Evaluation of One-on-One Hardware Synchronization Mechanism

Kiyoshi Hayakawa　Hitoshi Masumura　Hiroki Honda

Depertment of Computer Science, Yamanashi University
4 Takeda Kouhu-shi Yamanashi 400, Japan

For the efficient execution of synchronization on Fine-grain parallel processing, hardware synchronization mechanisms are needed. But, some hardware synchronization mechanisms produce an unnecessary wait time, when a synchronization guarantees the precedance constraints between tasks. In this paper, we propose One-on-One hardware synchronization mechanism that has no unnecessary wait time, and we evaluate the performance of One-on-One hardware synchronization mechanism which is implemented on the experimental MIMD machine.

1 はじめに

細粒度並列処理を効率良く行なうには、同期処理時間が細粒度タスクの実行時間と比べてかなり小さくなければならない。同期にかかる時間を少なくするため、同期処理のハードウェア化を行ない、同期処理のオーバーヘッドを減らすことが提案されている[1][3][4][5]。ハードウェア化を行なう同期モデルの1つとして、比較的動作が単純なバリア同期が挙げられる。

細粒度並列処理で行われる同期のほとんどが、先行制約を保証するための順序同期である。この順序同期を単純なバリア同期で行なうと、不必要な待ち時間が生じてしまい、効率良い並列処理が望めない。

不必要な待ち時間を低減させるため、バリアの概念を一部拡張させたハードウェア同期機構が提案されている[1][3][4][5]。その1つとしてSBMハードウェア同期機構[1]が知られている。SBM同期機構は、任意のプロセッサ間でのバリア同期を可能としている。

SBM同期機構で同期を行なうことにより、不必要な待ち時間は低減されるが、先行タスク実行後に不必要な待ち時間が残る場合がある。そこで、我々は先行タスク実行後の不必要な待ち時間が少ないOne-PE同期方式を提案した[7]。

One-PE同期方式では、タスクの実行時間変動がある範囲内におさまっていれば、不必要な待ち時間が無く先行制約を満たすことができる。しかし、その範囲を越えたタスク変動が生じた場合、One-PE同期でも、不必要な待ち時間が生じてしまう。

本稿では、タスクの実行時間変動が生じた場合においても不必要な待ち時間が無く、先行制約を満たすことができる1対1ハードウェア同期機構を提案する。また、1対1ハードウェア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なったので報告する。

2 不必要な待ち時間が無く先行制約を満たす同期方式

不必要な待ち時間が無く先行制約を満たす同期方法として、共有メモリに設けたフラグを用いた同期方式（以後、ソフトウェア同期方式と呼ぶ）が知られている。動作を以下に示す。

1. 先行タスクを実行しているPEは、先行タスクが終了した後、共有メモリに設けたフラグをセットする。

2. 後続タスクを実行しているPEは、後続タスクの実行前にフラグがセットされているか検査する。フラグがセットされている場合、後続タスクの実行に移り、セットされていない場合フラグがセットされるまで検査を繰り返す。

3 1対1ハードウェア同期機構の構成法及び動作

図1にPE4台における1対1ハードウェア同期機構の構成を示す。

先行制約を満たすためには、後続タスクを実行するPEは、先行タスクの実行終了を知る必要がある。ソフトウェア同期では、共有メモリに用意されたフラグにより、先行タスクの実行終了を知ることができるようにする。

1対1ハードウェア同期機構では、フラグのかわりにカウンタを用いて先行タスクの実行終了を知ることができる。

各PEには、他の全てのPEにおける先行タスク実行終了状況を知るために、他のPEに対応するカウンタが用意されている。（例えば、全PE台数N台の場合、各PEにN-1台のカウンタが用意されている。）先行タスクは、先行タスク実行終了後にカウンタをカウントアップさせる。後続タスクは、そのカウント数で各PEのどの先行タスクまでが実行終了しているかを知ることができる。つまり、後続タスクに対応した先行タスクの実行終了までに送出されるカウントアップパルスの数(N_p)とカウンタ値(N_c)を比較し、N_cがN_p以上なら後続タスクに対応した先行タスクの実行が終了されていることになる。

この比較を行なうために、comp回路を設けた。comp回路では、MPUから送られてくるデータとカウンタの値を比較し、カウンタの値がMPUから送られてくるデータ以上であれば、それを知らせるgon信号($n = 0,1,2,3$)を送出する。ここでMPUが送るデータは、N_pである。

selecter回路では、selecter信号を基に3つのgon信号の中から後続タスクに対応した先行タスクの実行終了を示すgon信号を選び、go信号としてMPUに知らせる。

同期コードには、カウンタインクリメントコードとカウンタチェックコードを用意する。

これら2種類の同期コードにおける挿入位置は、ソフトウェア同期と同じであるが、同期コードの動作が異なる。

カウンタインクリメントコードは、図1のup信号を出力させ、up信号につながっているカウンタをカウントアップさせ、次のタスクの実行に移る。例えば、PE0のup信号が出力された場合、PE1・2・3のCount 0がカウントアップされる。

カウンタチェックコードは、selecter信号及びcomp回路に送るN_pを送出し、go信号が返ってくるまで待つ。

当然、カウンタの値は有限なので、カウンタが数え切れなくなる前に、END信号を出力しバリア同期を行ない、カウンタの値を0にする。

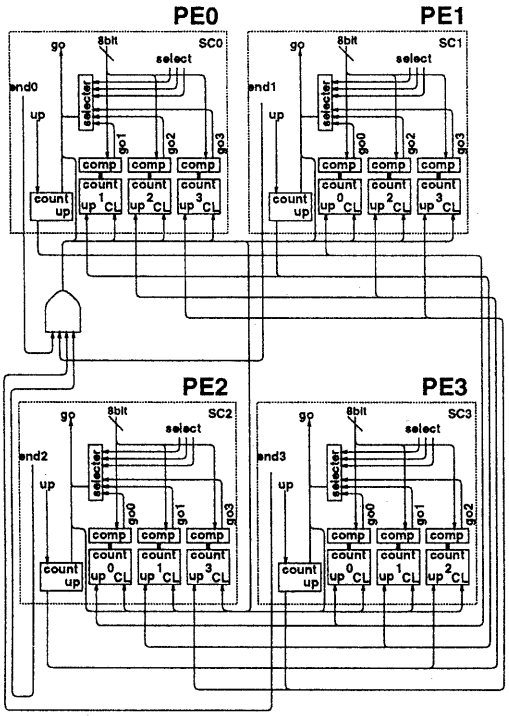

図 1: 1対1ハードウェア同期機構の構成

4 1対1ハードウェア同期機構の特徴

4.1 カウンタインクリメントコードにおける動作の特徴

1対1同期におけるカウンタインクリメントコードの動作では、先行制約に関係のないPEのカウンタにもCount-up信号を送出する。よって、カウンタの値には、無駄なカウントも含まれているが、処理のオーバーヘッドとなることはない。また、1対1同期において、後続タスクが他の全てのPEに存在した場合、1本のConut-up信号線にカウントパルスを送出すれば、他の全てのPEのカウンタをアップでき、他の全てのPEに先行タスクの実行終了を知らせることができる。

4.2 カウンタチェックコードにおける動作の特徴

1対1同期の場合、カウンタコード1命令に対して1つの比較器の情報しか参照できない。よって、1つの先行タスクに対し1つの後続タスクの場合におけるカウンタチェックコードは、1命令でよいが、複数の先行タスクに対して1つの後続タスクの場合におけるカウンタチェックコードは、先行タスク数と同じ命令数必要となってしまう。

5 性能評価

1対1ハードウェア同期機構の有効性を調べるために、実マルチプロセッサ上で性能評価を行なった。

5.1 OPAS システム

実マルチプロセッサ上で性能評価を行なうために、共有メモリ型MIMDマルチプロセッサシステムOPASを開発した。

5.2 OPAS システムのアーキテクチャ

図2にOPASのアーキテクチャ構成を示す。OPASは、4台のPE(Processing Element)とShared Memoryを、1本のVMEバスで結合した共有メモリ型マルチプロセッサである。

各PEは、MPU(MC68HC000(8MHz))と256KByteのローカルメモリで構成される。Shared Memoryは、1MByteのS-RAMで構成される。Timerは、実行時間計測のタイマ及びタイマ割り込みのためのタイマで構成されている。

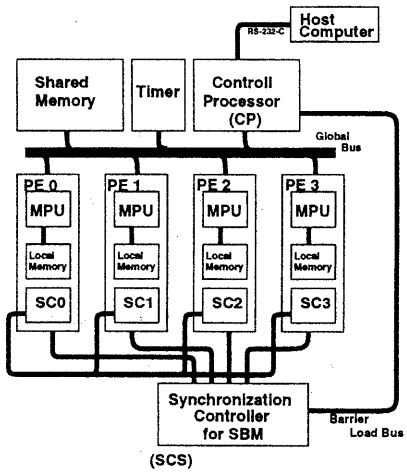

図 2: OPAS システムのハードウェア構成

Control Processor(CP)は、ホストコンピュータで生成された並列オブジェクトプログラムを各プロセッサにダウンロードし、実行結果等をホストコンピュータに送る。また、CPは、Barrier Load Busを介してSynchronization Controller(SC)のBQにバリア参加情報を挿入する。

Synchronization Contoroller (SC)は、SBM同期機構用のSC (SCS)と1対1ハードウェア同期機構用のSCn ($n = 0..3$)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、SBMバリ

ア同期成立の検出・PE Go 信号の送出を行なう。SCn は、お互いに up 信号線で結合され、カウンタの up 信号の送出、先行タスクの実行終了の検出を行なう。

5.3 1対1ハードウェア同期機構のインプリメント

図3に1対1ハードウェア同期機構の回路図を示す。Local Address Bus は 24 ビット、Synchronization Data Bus は、11 ビットのバス幅である。カウンタは、非同期カウンタの 74393 で 8 ビットカウンタで構成した。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 信号の送出を行なう。Selector は、SEL 信号で指定された Gon 信号 ($n = 0, 1, 2$) の1つを Go 信号として送出させる。

5.3.1 カウンタインクリメントコード動作の実現

カウンタインクリメントコードの実行時に、count up 信号を送出しなければならない。この Count up 信号送出するために、メモリアクセスを利用する。以下に、メモリアクセスを利用した Count-up 信号の送出動作を示す。

1. 各 PE 内の MPU が FFF002 番地にアクセスをする。
2. Decoder 回路で FFF002 番地のアクセスを検出したら、Conut-up 信号を送出すると同時に Go 信号も送出する。
3. Go 信号を各 MPU の DTACK* 信号とし、メモリの代わりに SCn が DTACK* 信号のアサートを行なう。

5.3.2 カウンタチェックコード動作の実現

カウンタチェックコード実行時は、先行タスクの実行が終了するまで待ち、先行タスクが実行終了したらただちに待ち状態を解除しなければならない。この同期待ち状態と待ち状態解除の実現は、MPU のメモリ書き込み動作を利用する。動作を以下に示す。

1. 各 PE 内の MPU が FFF004 番地にカウンタのセレクト番号及び先行タスクにおける実行終了時点でのカウンタの値を示す情報を書き込む。
2. 先行タスクの実行終了時点のカウントアップパルス数 N_p 及びどの比較器の情報を有効にするのかという情報（SEL0 ～ SEL2）を Synchronization Data Bus に流す。
3. Synchronization Data Bus のデータが値が定まり、比較器の判定信号（$P = Q$, $P > Q$ ）の値が定まったら、SEL 信号で指定した Gon($n = 0, 1, 2$) 信号を Go 信号にする。
4. Go 信号を各 MPU の DTACK* 信号とし、メモリの代わりに SCn が DTACK* 信号のアサートを行なう。

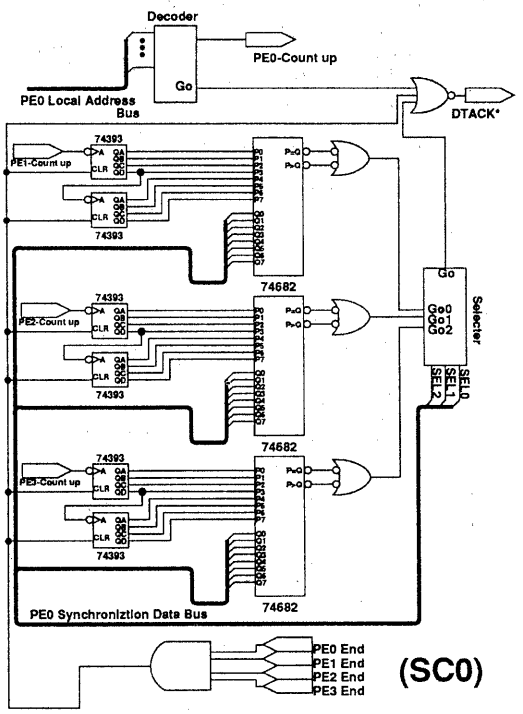

図 3: 1対1ハードウェア同期機構の回路図（PE0）

5.4 性能評価した同期方式

性能評価は、テストプログラムを四種類の同期方式で実行し、実行時間を比較した。同期方式は、次の四種類である。One-PE・SBM・ソフトウェア同期は、1対1同期との比較のために行なった。

- 1対1同期方式
- One-PE 同期方式
- SBM 同期方式
- ソフトウェア同期方式

これら4種類の同期方式の同期コードの挿入位置は、先行タスク実行直後と後続タスクの実行開始直前に挿入することとする。

5.4.1 各同期方式の説明

SBM ハードウェア同期機構を用いた SBM 同期及び One-PE 同期の同期コード挿入及び動作を説明する。

SBM 同期方式　先行タスク実行終了直後と後続タスクの実行開始直前に同期コードを挿入しSBM 同期を行なう場合、その同期の付加的先行制約から生じる冗長な同期を除去し、SBM 同期を行なうことを、SBM 同期方式と呼ぶ [7] 。

例えば、図 4の矢印で示すような Task1 と Task10、Task2 と Task9 に先行制約が存在した場合、図 5のように Task1 の後と Task10 の前にバリアコードを挿入し、バリア参加情報を 1100 としてSBM 同期を行なう。この同期の付加的先行制約により、Task2 と Task9 に対する先行制約が満たされるので、Task2 と Task9 に対する同期コードやバリア参加情報をプログラムや Barrier Queue に挿入する必要がない。

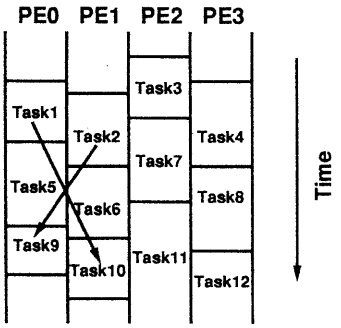

図 4: 冗長な同期が生じる先行制約の例

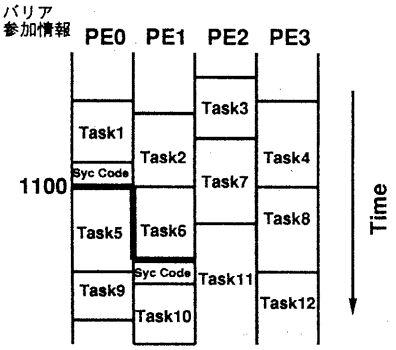

図 5: 図 4の先行制約を満たす同期コードの挿入

One-PE 同期方式　先行タスク実行後の不必要な待ち時間を少なくするために、BQ によるバリア実行の順づけ（ブロッキング）から生じる先行制約を利用し同期を行なうこと One-PE 同期方式という [7]。

例えば、図 6のような先行制約を満たすことを考える。SBM 同期では、この先行制約を満たすために、先行タスクと後続タスクの参加情報を 1 つのバリア参加情報として BQ に挿入する。一方、One-PE 同期では、この先行タスク・後続タスクのバリア参加情報をプロセッサごとに分割し、先行タスクが参加しているのバリア参加情報を先、後続タスクの参加しているバリア参加情報を後、の順で BQ に挿入する（図 6参照）。このように挿入すれば、ブロッキングにより後続タスクの実行は必ず先行タスクの後とすることができる。さらに先行タスクを実行するプロセッサはその先行タスクの終了後、1 台で同期を行ない次の実行を進めることができるので、先行タスク実行終了後の不必要な待ち時間がない（図 7参照）。図 6の例では、同期コードの変動が生じた場合でも先行タスクの実行終了後に不必要な待ちは生じない。

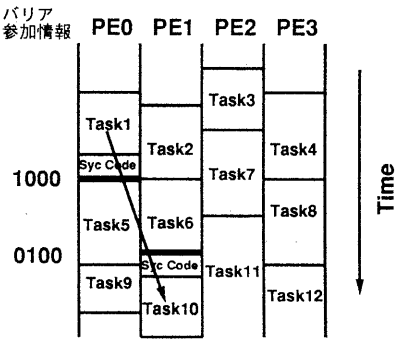

図 6: One-PE 同期方式の同期コード挿入例

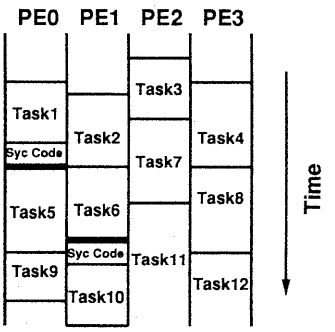

図 7: 図 6における実際の実行タイミング

5.5　テストプログラムの記述言語

OPAS システムで実行したテストプログラムは、C 言語で記述した。この C のソースプログラムを本研究室で開発した 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。

1. C の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. 分割したタスクのタスクグラフを生成する。
3. タスクグラフを基に CP/MISF 法 [9] を用いたタスクスケジューリングを行ない、実行前に各 PE で実行するタスクを決める。

4. 各PEにスケジューリングされたタスクの先行制約にしたがって同期コードを挿入する。

5.6 性能評価1

性能評価1として、差分法による流体のシミュレーションを行なうプログラムの一部を並列実行した。このテストプログラムの同期が必要な先行制約を図8・9に実行結果を表1に、PE台数と速度向上比(=PE 1台のSBM同期における実行時間/各々のPE台数の各同期方式における実行時間)の関係を図10に示す。図8・9において、丸印がタスク、直線が先行制約を表し、左から右へタスクの実行が行なわれる。また、速度向上比は、PE1台のSBM同期における処理速度を1とした場合の比率である。

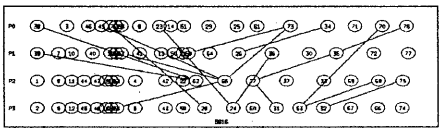

図 8: 評価プログラム1の同期が必要な先行制約(SBM同期)

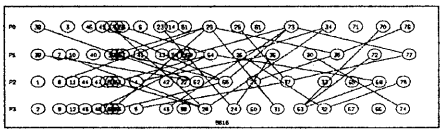

図 9: 評価プログラム1の同期が必要な先行制約(1対1同期・One-PE・ソフトウェア同期)

表 1: 評価プログラム1の実行結果

PE台数	速度向上比			
	SBM	One-PE	Soft	1対1
1	1	1	1	1
2	1.58	1.85	1.74	1.9
3	1.60	2.2	2.00	2.4
4	1.75	2.71	2.0	2.62

5.6.1 評価プログラム1の実行結果の考察

One-PE同期と1対1同期は、良好な結果を得た。わずかではあるが、PE2・3台では1対1同期の方が速く、PE4台ではOne-PE同期の方が速い。これには、次の2つの原因が考えられる。

- 1対1同期の消費者同期コードの同期処理がOne-PE同期より最悪で2クロック遅い。
- 第二に、複数の先行タスクに対して1つ後続タスクの場合におけるカウンタチェックコードの命令数が、One-PE同期方式より多い。

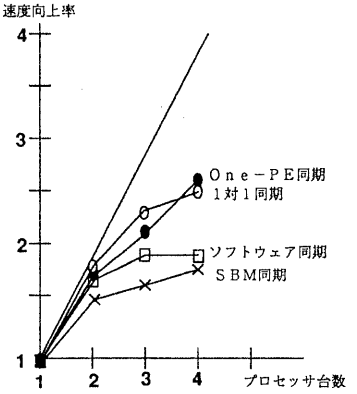

図 10: 評価プログラム1における速度向上率とPE台数の関係

例えば、図11・12の矢印で示す先行制約を満たす同期コードを考える。One-PE同期の場合、図11のように4つ同期コードが必要である。1対1同期の場合は、図12のように6つの同期コードが必要であり、One-PE同期と比べると2つ多い。この原因は、PE台数に比例して増加する。

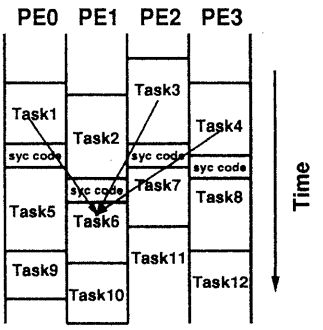

図 11: 先行タスクが複数存在する先行制約の同期コード挿入(One-PE同期)

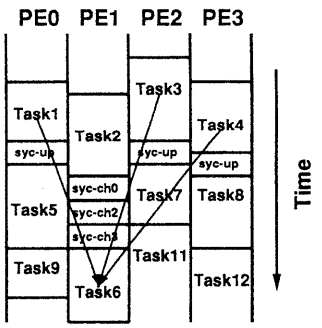

図 12: 先行タスクが複数存在する先行制約の同期コード挿入(1対1同期)

5.7 性能評価2

性能評価2として、ブロッキングによる不必要な待ちによる影響を調べるために、性能評価1で実行したプログラムのタスク実行時間を意図的に変動させたプログラムをPE4台で並列実行し、性能評価を行なう。

評価プログラム2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1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プログラムを付け加える。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。

以上の方法を用いて、10の評価プログラムを作成した。

各評価プログラムの変動タスク数と平均NOP数及び実行時間比を表2・3に示す。また、各同期方式における実行時間比の度数分布を図13・14・15に示す。なお、実行時間比は、次の式で表される。

$$\text{実行時間比} = \frac{\text{各同期方式における並列実行時間}}{\text{1対1同期における並列実行時間}}$$

5.7.1 性能評価2の実行結果の考察

表2・3から、全ての場合において1対1同期が良好な結果を示していることがわかる。

また、平均NOP数が30未満のOne-PE同期方式における実行時間比は、全て1.1未満である。一方、平均NOP数が30以上のOne-PE同期方式における実行時間比は、1つを除いて1.1以上になり、平均NOP数が最大であるNo.3の評価プログラムでは、One-PE同期はソフトウェア同期よりも遅くなってしまう。以上のことから、One-PE同期は、タスクの実行時間変動が許容範囲を越えれば越えるほど、ブロッキングにより実行時間がより遅くなることがわかる。

図13・14・15でわかるように、One-PE同期方式は、ブロッキングにより実行時間が遅くなるものの、1つを除いてソフトウェア同期より性能が良い。SBM同期は、先行タスク実行後の不必要な待ち時間及びブロッキングによる待ち時間が原因で、最も性能が悪くなってしまう。

6 おわりに

1対1ハードウェア同期機構を提案し、実マルチプロセッサ上で性能評価を行なった。その結果、以下のことがわかった。

- タスクに実行時間変動がある場合、1対1同期が向いている。
- タスクの実行時間変動が少なく1つの後続タスクに対し複数の先行タスクが存在するプログラムでは、One-PE同期方式が向いている。

今後の課題として、以下のことが挙げられる。

- 先行タスクが複数存在した場合のカウンタチェックコードの命令数を少なくするアーキテクチャを検討する。
- 1つの後続タスクに対し複数の先行タスクが存在する場合の同期はOne-PE同期で行ない、1つの後続タスクに対して1つの先行タスクが存在する場合の同期は、1対1同期で行なうというような、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を用いて性能評価を行ない、広範囲にわたる1対1同期の有効性を検証する。

表2: 各評価プログラムの変動タスク数と平均NOP数

No.	変動タスク数	平均NOP数
1	11	33.3
2	24	29.0
3	8	41.6
4	7	36.1
5	12	31.8
6	22	26.3
7	18	27.7
8	6	17.8
9	20	27.4
10	12	20.2

表3: 評価プログラム2の実行時間比（PE4台）

No.	実行時間比			
	SBM	One-PE	Soft	1対1
1	1.49	1.13	1.26	1
2	1.52	1.08	1.20	1
3	1.73	1.21	1.20	1
4	1.84	1.15	1.24	1
5	1.70	1.17	1.27	1
6	1.74	1.04	1.16	1
7	1.87	1.06	1.18	1
8	1.69	1.05	1.29	1
9	1.52	1.11	1.25	1
10	1.75	1.08	1.22	1

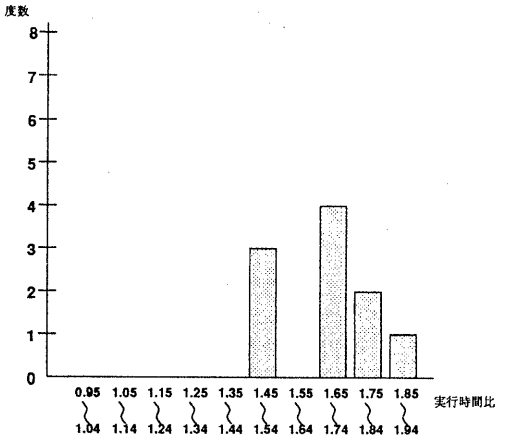

図 13: 実行時間比の度数分布（SBM 同期）

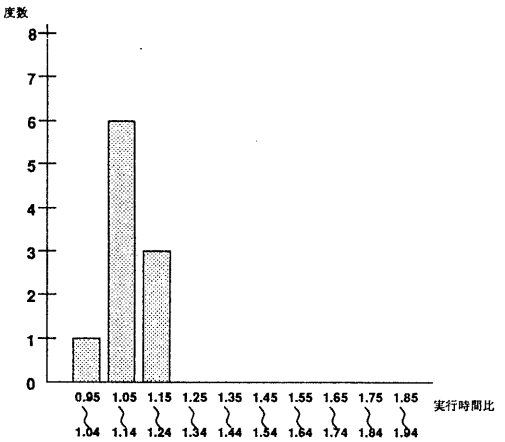

図 14: 実行時間比の度数分布（One-PE 同期）

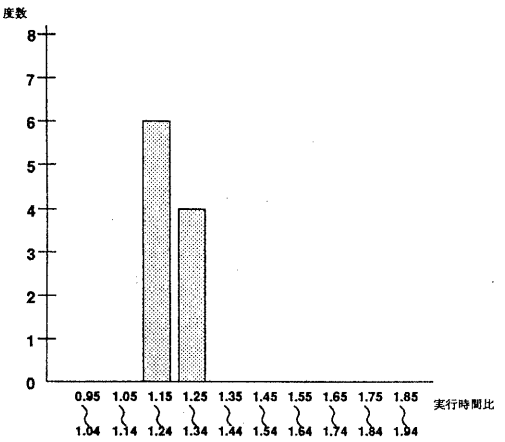

図 15: 実行時間比の度数分布（ソフトウェア同期）

なお本研究の一部は、文部省科学研究費補助金奨励研究 (A)05750336 による。

参考文献

[1] O'Keefe,M.T.and Dietz,H.G."Hardware Barrier Synchronization: Static Barrier MIMD(SBM)," *proc. Int'l Conf. Parallel Processing.* Vol.I, pp.35-42, Aug.1990.

[2] Gupta, R., "The Fuzzy Barrier: A Mechanizm for High Speed Synchronization of Processors." *Proc. 3rd Int'l. Conf. Architectural Support for Programming Languages and Operating System,* pp.54-63m Apr.1989.

[3] 松本尚, "細粒度並列実行支援マルチプロセッサの検討" 情報処理学会論文誌. Vol31 No12, pp.1840-1851,Dec.1990

[4] 高木, 有田, 曽和, "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* 論文集, pp269-276, May.1991

[5] 高木, 有田, 曽和, "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, pp1583-1591

[6] 高木, 有田, 川口, 曽和 "バリアを唯一の同期手段とした場合のタスクスケジューリング" 電子情報通信学会技術研究報告 CPSY93− 22, P73-80, Aug, 1993

[7] 増村, 早川, 本多, "SBM 同期ハードウェアを用いた同期方式の検討" 電子情報通信学会技術研究報告 CPSY93-21, P65-72,Aug, 1993

[8] 早川, 増村, 本多, "SBM 同期機構の実装方法と性能評価" 電子情報通信学会技術" 研究報告 CPSY93-20, P57-64,Aug, 1993

[9] H.Kasahara and S.Narita, "Practical Multiprocesssor Scheduling Algorithms for Efficient Parallel Processing." *IEEE Trans. Comput.,* Vol.C33, No.11, pp.1023-1029, Nov.1984