計算機アーキテクチャ 125－4
（1997. 8. 20）

ブロッキングを利用した同期方式における冗長同期コード削減手法の検討

早川 潔† 本多 弘樹‡

†山梨大学電子情報工学科
‡電気通信大学大学院情報システム学研究科

概 要 近細粒度並列処理で,先行制約を保証する同期を,高速に且つ不必要な待ち時間なく行うための同期機構・方式のひとつとして,バリアキューによるブロッキングを利用して先行制約を保証するRBCQ同期機構とその同期方式が提案されている.RBCQ同期方式による同期コード配置では,冗長な同期コードが配置されてしまう.そこで,本稿では,RBCQ同期方式で生じる冗長同期コードの削減手法を検討する.次に,強制参加でブロッキングを利用することにより,バリアキューがない同期機構として,RBNQ同期機構／同期方式を提案する.RBNQ同期方式では,RBCQ同期方式に比べ,同期コードが多くなってしまう.そこで,各プロセッサ内に回路を組み込むことにより,同期コード数を低減する機構を検討する.

Examination of Method to Reduce The Redundant Synchronization Codes for The Synchronization Method Using Barrier Blocking

Kiyoshi Hayakawa † Hiroki Honda ‡

† Department of Electrical Engineering and Computer Science, Yamanashi University
‡ Graduate School of Information Systems University of Electro-Communications

Abstract Several hardware synchronization mechanisms have been proposed to reduce synchronization overhead on fine-grain parallel processing. Firstly, we examine the methods to reduce the redundant codes for the RBCQ method. We have proposed the RBCQ synchronization mechanism/method. When the RBCQ synchronization method produces the synchronization codes, The redundant synchronization codes are included among these codes. Secondly, we describe the RBNQ synchronization mechanism/method. The RBNQ synchronization method behaves as if a Barrier model that all processors participate in the surface barrier. The RBNQ synchronization method is more synchronization codes than the RBCQ synchronization method. We describe the circuit that allows to reduce the synchronization codes.

1 はじめに

密に結合されたマルチプロセッサのプロセッサ間同期オーバーヘッドを少なくするために, 種々のハードウェア同期機構・方式[1]が提案されている [1][2][3][9][7][6]. これらの同期機構の多くは, バリア同期を一部拡張した同期動作を行う同期機構（バリア型同期機構）である. 本稿では, 近細粒度並列処理で, タスク間の先行制約を保証するためのバリア型同期機構について議論する.

バリア型同期機構の同期ポイントの指定方法として, 命令フィールド中に同期情報を示すタグを設ける方法と同期情報を示す専用命令（同期コード）を並列化オブジェクトコード内に配置する方法が考えられている [3].

マルチプロセッサを汎用プロセッサで構成する場合, 同期コードによる方法のほうがタグによる方法より簡単に構成できる. 同期コードによる方法の場合, 同期機構依存先行制約 [6] による不必要な待ち時間の削減とともに同期コード数の削減も重要となる [4].

同期機構依存先行制約による不必要な待ち時間および同期コード数が少ない同期機構として, RBCQ 同期機構/同期方式が提案されている [10]. しかし, このRBCQ 同期方式には, 冗長な同期コードが配置されてしまう. そこで,RBCQ 同期方式で配置される冗長同期コードの削減手法を検討する.

次に, 強制参加でブロッキングを利用することにより同期を行う同期機構/同期方式として,RBNQ[2]同期機構/同期方式を提案する. この同期方式では, 全てのプロセッサが同期に参加し, ブロッキングを利用した同期を行うので,RBCQ 同期方式より同期コード数が多い. そこで, 同期コードを工夫することにより,RBNQ 同期方式での同期コード数を削減する方法を検討する.

なお, 本稿では以下の事項を前提とする.

- ターゲットマシン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とする.
- 汎用プロセッサでのインプリメントの容易さを考え, 同期コードによる同期ポイント指定方式を採用する.
- 並列処理方式は, 逐次プログラムの基本ブロックを近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 [8] するものとする.
- 並列化コンパイラが, コンパイル時に, 各プロセッサ用のオブジェクトコードを生成する.
- 見積り実行時間と実際の実行時間のずれ（タスクの実行時間変動）が生じる場合がある. だだし, タスクの実行時間変動は比較的短い時間（1〜2 タスクの実行時間）とする.

[1]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先行制約を保証するための方法を意味する.

[2]Reduced barrier Blocking Non barrier Que の略

2 RBCQ 同期機構/同期方式

2.1 RBCQ 同期機構の動作と同期検出回路

RBCQ 同期機構は「ある同期に参加しているプロセッサは, 直前の同期に参加している全てのプロセッサが同期ポイントを通過しなければ, 同期ポイントを通過することはできない」という動作を行う.

上記の動作を実現する RBCQ 同期機構の同期処理回路を図 1 に示す. プロセッサは, 同期コードを実行する際,PE Wait 信号を出し,PE Go 信号が出されるまで, 待ち状態となる.RBCQ 同期機構の同期処理回路は, バリアキューおよび同期成立検出回路で構成される. 同期成立検出回路内では, 以下の処理が行われる.

- キューの先頭（BQ-Head）のバリア参加情報 [9] とプロセッサが送出する PE Wait 信号との AND を PE Go 信号とする.
- これと同時に,BQ-Head に格納されているバリア参加情報内で,PE Go 信号が送出されたプロセッサのビットのみを, 同期成立検出回路がクリアする.
- BQ-Head の内容が全て 0 になった時点で Next 信号を出力し, バリアキューを右にシフトさせ, 次のバリア参加情報をキューの先頭に設定する.

2.2 RBCQ 同期方式

RBCQ 同期機構を使用した先行制約保証のための同期コード配置決定方法およびバリア参加情報列生成方法を示す.

同期コード配置は,先行タスクの直後および後続タスクの直前とする. また,各同期コードの見積り実行開始時刻をそれぞれ先行タスクの見積り実行終了時刻と後続タスクの見積り実行開始時刻とする.

バリア参加情報列の生成手法では,同期コードをグループ化し,各グループ間の順序付けを行う.それを基にして,バリア参加情報列の生成を行う.

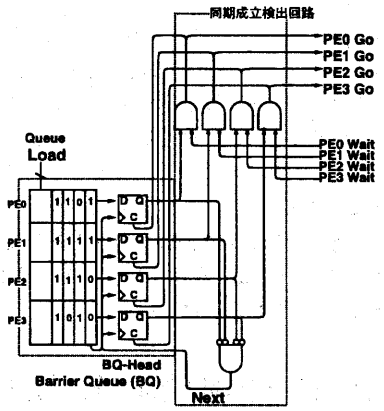

図 1: RBCQ 同期機構の回路（PE4 台）

同期コードのグループ $S_i(i=1,..,n)$(n はグループ数）は, 以下の条件の下で, 各グループ内の同期コード数が最大になるように構成する.

- 同一グループ内には,同じプロセッサで実行される同期コードは存在しない.
- 同一グループ内の各同期コードに対応するタスク間に,先行制約がない.
- グループ S_i内の同期コードの見積り実行開始時刻の集合を T_iとすると,T_iと T_{i+1}の間には,$maxT_i \leq minT_{i+1}$ の関係が成り立つ. ただし,$maxT_i = minT_{i+1}$ で, $maxT_i$の同期コードに対応したタスクと,$minT_{i+1}$の同期コードに対応したタスクとの間に先行制約がある場合,先行タスクに対応した同期コードは S_iのグループに属し,後続タスクに対応した同期コードは S_{i+1}のグループに属している.

S_i内の同期コードを実行するプロセッサのグループを P_iとし, P_i内のプロセッサが参加するバリア参加情報を生成し,それらを i の値が小さい順に列べ,バリア参加情報列とする.

先行タスクが属しているグループを S_iとすると,上記の条件により,その先行タスクと先行制約の関係にある後続タスクは,必ず S_{i+1}以降のグループに属する.なぜなら,各タスクはリストスケジューリングを用いてスケジューリングされているので,後続タスクの見積り実行開始時刻は先行タスクの見積り実行終了時刻以降にスケジュールされているからである.

2.3 RBCQ 同期方式の冗長な同期コード

RBCQ 同期方式において,以下の条件を満たす場合,冗長な同期コードが存在し,その同期コードを削除しても問題が持つ先行制約を保証できる.

条件 先行タスクに対応する同期コードを p, 後続タスクに対応する同期コードを s とし,p,s が属する同期コードグループをそれぞれ $G(p)$,$G(s)$ とする. また,p と s 以外の同期コードを d,d が属する同期コードグループを $G(d)$ とする.

$G(p)$ と $G(s)$ の間に $G(d)$ が存在し,d が,p と同じプロセッサで実行されるのであれば,pが冗長な同期コードであり, s と同じプロセッサで実行されるのであれば,s が冗長な同期コードである.

上記の条件を満した場合、冗長な同期コードと dとの間で逐次実行による同期コードの順序づけが生じ,d と冗長でない同期コードとの間でブロッキングによる同期コードの順序づけ生じる. これらの同期コードの順序づけによって,p,s に対応するタスク間の先行制約が満たされる.

もし,冗長な同期コードを削除した場合,d に対応したタスクと冗長でない同期コードに対応したタスク間の先行制約（以後, ブロッキング先行制約と呼ぶ）が, 同期の必要な先行制約となる.

例えば, 図 2の Task1 と Task16 の先行制約の場合, Task1 直後の同期コードが p,Task16 直前の同期コードが s となる. また,Task9 の直前の同期コードが d となり,d と同じプロセッサで実行される p が冗長な同期コードとなる.

もし,p を削除した場合,d に対応したタスクと s に対応したタスク間にブロッキング先行制約が生じる.

2.4 冗長コードの削減アルゴリズム

冗長な同期コードを削除するアルゴリズムの単純な方法として、以下のアルゴリズムが挙げられる。

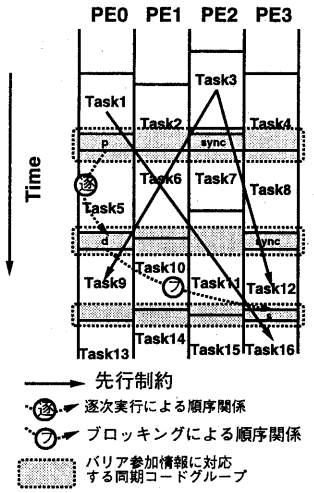

図 2: RBCQ 同期方式の冗長な同期コードの例

1. 文献 [10] の手順にしたがい,RBCQ 同期方式の同期コード配置とバリア参加情報列の生成を行う.

2. 先行タスク直後の同期コードの見積もり実行時刻が早い順にその先行タスクの同期コード p を 1 つずつ選ぶ.

3. 2. で選んだ p に対応する後続タスクの中で,見積り実行開始時刻の一番早い後続タスク直前の同期コード s を求め,それらの同期コードが属している同期コードグループ $G(p)$,$G(s)$ を求める.また,p に対応するタスクとの間でブロッキング先行制約が生じているタスクの同期コード $q_n(n = 1..i)$(i はブロッキング先行制約の数)を求め,それらの同期コードが属している同期コードグループ $G(q_n)$ を求める.

4. $G(p)$-$G(s)$, $G(p)$-$G(q_1)$,..,$G(p)$-$G(q_i)$ の全ての同期コードグループ対の間に存在し,且つ,p と同じプロセッサで実行される同期コード d をもつ $G(d)$ を求める.求めることができたら,p を削除し,その同期コードに対応するバリア参加情報内のビットを 0 にし,d に対応するタスクと s に対応するタスク間にブロッキング先行制約が生じていることを示すマークをつけ,2. へもどる.そうでなければ,次へ.

5. 3. で求めた s に対応するタスクとの間でブロッキング先行制約が生じているタスクの同期コード $t_m(m = 1..j)$(jはブロッキング先行制約の数)を求め,それらの同期コードが属している同期コードグループ $G(t_m)$ を求める.

6. $G(p)$-$G(s)$,$G(s)$-$G(t_1)$,..,$G(s)$-$G(t_j)$ の全ての同期コードグループ対の間に存在し,且つ,s と同じプロセッサで実行される同期コード d' をもつ $G(d')$ を求める.求めることができたら,s を削除し,その同期コードに対応するバリア参加情報内のビットを 0 にし,d' に対応するタスクと p に対応するタスク間にブロッキング先行制約が生じていることを示すマークをつけ,2. へもどる.そうでなければ,次へ.

3 RBNQ 同期機構/同期方式

3.1 RBNQ 同期機構の動作と同期検出回路

RBCQ 同期機構では,任意参加によるブロッキングを利用して同期を行っていた.その任意参加を実現するために,バリアキューが必要となる.このバリアキューがあることにより,以下の問題が生じる.

1. バリア参加情報を再投入する場合に生じるオーバーヘッド.

2. キュー回路のハードウェアコスト.

1. の問題を解決するために,バリアキューを複数用意し,同期操作とバリア参加情報の再投入操作をオーバーラップさせることによりオーバーヘッドを隠蔽できると考えられる.しかし,2. の問題は依然として解決されないままである.そこで,このバリアキューのハードウェアコストを削減するために,強制参加によるブロッキングを利用し,同期を行う同期機構(RBNQ 同期機構)とその同期方式を提案する.

RBNQ 同期機構では,「面状&強制参加&全順序関係&オーバーラップ不可能 [7]」なバリア同期モデルの出口のバリア領域と入口のバリア領域が接しているような動作を行う.つまり,「ある同期ポイントに到達したプロセッサは,全てのプロセッサが直前

の同期ポイントを通過しなければ, その同期ポイントを通過することはできない」という動作を行う.

RBNQ 同期機構のプロセッサ 4 台での構成を図3に示す. RBNQ 同期機構は, 同期検出回路と BQ-Head のみで構成される. RBNQ 同期機構の同期検出回路は,RBCQ 同期機構と同じ回路であるが, バリアキューの回路が異なる.RBNQ 同期機構では,Next 信号が送出されると,BQ-Head 内の全てのフリップフロップをプリセット (“1”) にする.

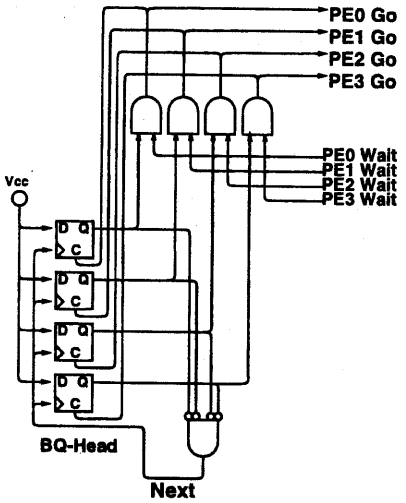

図 3: RBNQ 同期機構の回路 (PE4 台)

3.2 RBNQ 同期方式

RBNQ 同期方式の同期コード配置の決定およびバリア参加情報列の生成手順を以下に述べる.

1. RBCQ 同期方式と同じ手順を用いて, 先行タスクの直後と後続タスクを直前に同期コードを配置し, 同期コードのグループ $S_i(i = 1, .., n)$(n はグループ数) を作成する.

2. 1. で求めた同期コードのグループ S_i を i が小さい順に 1 つずつ選択する.

3. 2. で選択された S_i 内の同期コードを実行しないプロセッサを求める.

4. 3. で求めたプロセッサにダミーの同期コードを配置する. 配置場所は, S_{i+1} のグループの中で, 一番早い同期コード見積り実行時刻に近い, タスクの見積り実行開始 (または終了) 時刻に配置する.

例えば, 図 4の矢印の先行制約を満たすために, RBCQ 同期方式では図 4に示す位置に同期コードを配置する.

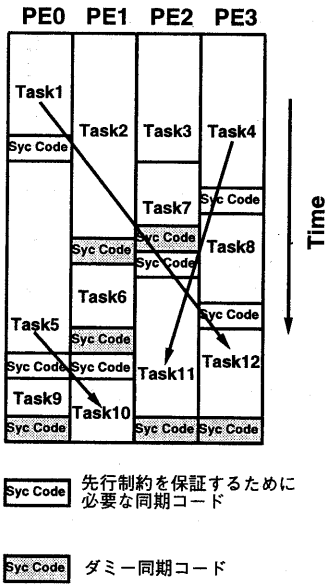

図 4: RBNQ 同期機構の同期コード配置例

4 RBNQ 同期方式の同期コード削減手法

RBNQ 同期機構を用いる場合, バリアキューのハードウェアコストを削減できるが,RBNQ 同期方式で生成される同期コード数が RBCQ 同期方式のそれより多くなってしまう. そこで, 連続する同期ポイントを 1 つの同期コードで処理できるように, 同期コードの実行時に行われる処理を改良する.

同期コードの実行時に行われる処理を改良するために, 各プロセッサに図 5に示す回路を組み込む.

プロセッサは, 同期コード実行時に, 連続している同期ポイント数を Data Inputs に入力し,PE Go (to MPU) 信号が送出されるまで待つ.Down カウンタの Data Outputs が 1 以上の間,PE Wait 信号を送出し続ける.Down Counter では,PE Go(from RBNQ) 信号入力されるたびに,Data Inputs から入力された値をデクリメントする.Down Counter の値

が 0 になったら,PE Wait 信号の送出を中止し, PE Go(to MPU) 信号を送出する.

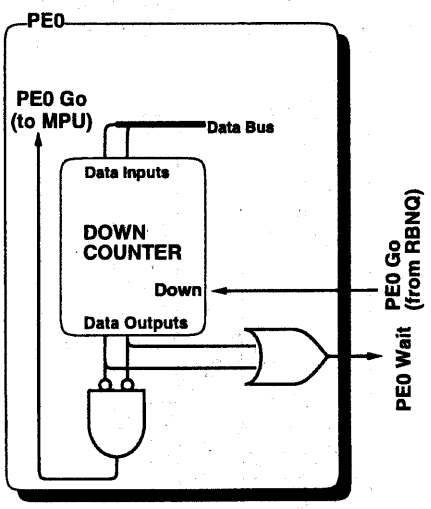

図 5: 同期コード動作を改良するための回路 (PE0)

RBNQ 同期方式で生成された同期コードを,改良した同期コードに置き換える。そのため、RBNQ 同期方式の同期コードを検索し,連続している同期コードを見つけた場合, それらの同期コード数を Data Inputs に出力する同期コードに置き換える. 他の同期コードは,"1" を Data Inputs に出力する同期コードに置き換える.

5 まとめ

RBCQ 同期方式による同期コード配置で生じる冗長な同期コードを削減する手法を検討した. また, 強制参加でブロッキングを利用することにより, バリアキューがない同期機構として,RBNQ 同期機構／同期方式を提案した. さらに, 同期コード数を低減する機構として, 同期コードでの処理を改良する機構を検討した.

今後の課題として, 本稿で検討した冗長同期コード削減手法が, どの程度最適に（不必要なブロッキングが少ない, 同期コード数を少ない, 同期コードグループ数が少ない）冗長な同期コードを削除しているか調べ, 冗長同期コード削減手法の最適化を検討する.

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 Mechanism for High Speed Synchronization of Processors. "*Proc. 3rd Int'l. Conf. Architectural Support for Programming Langua-ges and Operating System,* pp.54-63 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. Vol.31 No.12,pp.1840-1851,Dec.1990.

[4] 稲垣 達氏, 松本 尚, 平木 敬, "システムの階層的並列性を統一的に扱う最適化コンパイラ", 電子情報通信学会技術研究報告 CPSY94-40,pp.105-112,Jul.1994.

[5] 高木 浩光, 有田 隆也, 川口 喜三男, 曽和 将容 "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, pp.73-80,Aug,1993.

[6] 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,pp.1583-1591,Dec.1991.

[7] 山家 陽, 村上 和彰,"バリア同期モデルー Taxonomy と新モデルの提案, および, モデル間性能比較ー" 並列処理シンポジウム *JSPP'93* 論文集,pp.119-126, May.1993.

[8] 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-1, No.9, pp.756-766, Sep.1990.

[9] 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, Feb.1995.

[10] 早川 潔, 本多 弘樹, "RBCQ 同期機構およびその同期方式の提案と性能評価", 並列処理シンポジウム *JSPP'97* 論文集,pp.45-52, May.1997.