システムソフトウェアと
オペレーティング・システム 70−15
（１９９５．８．２３）

細粒度並列処理のためのハイブリッドスケジューリングシステム：スケジューラによるバリア同期管理手法

松田 孝史　武石 尊之　野口 善昭　岩根 雅彦
matsuda@arch.comp.kyutech.ac.jp
九州工業大学工学部

共有メモリマルチプロセッサシステム上での細粒度並列性を利用した並列処理においてレスポンスタイムとスループットを両立することを目的とした、静的／動的スケジューリングを組み合わせたハイブリッドスケジューリングシステムを提案する。プログラムは粗粒度の並列性などから複数のスレッドに分割され、各スレッドは細粒度の並列性を利用してコンパイラにより静的にスケジューリングされる。動的スケジューラは同時に複数のプロセッサを１つのスレッドに割り当てる。各プロセッサ協調して動作し、バリア同期によって同期をとる。バリア同期をとるプロセッサ群のバリア同期機構への登録は動的スケジューラがプロセッサ割り当てと併せて行う。本システムを細粒度並列計算機 MSBM 上で Whetstone ベンチマークを用いて評価した。

Hybrid Scheduling System for Fine Grain Parallelism: Scheduling Method united with Barrier Management

Takashi Matsuda , Takayuki Takeishi , Yoshiaki Noguchi , and Masahiko Iwane

Faculty of Engineering, Kyushu Institute of Technology

This paper proposes a hybrid scheduling system, a combination of static and dynamic scheduling. The objective of the system is not only to reduce the response time but also to increase the throughput when fine grain parallelism is utilized. A compiler partitions a program into a number of threads based on coarse grain parallelism, control flow, and so on. Then it schedules each thread statically based on fine grain parallelism. The dynamic scheduler dispatches one thread to the required number of processors and registers the processors which synchronizes with each other to barrier synchronization mechanism. The effect of the scheduling system is evaluated using Whetstone benchmark on MSBM computer for fine grain tasks.

1 はじめに

半導体集積回路技術の向上により、複数の中央演算処理装置を単一のチップ上に集積するマルチプロセッサオンチップが技術的および採算的に可能になりつつある。マルチプロセッサオンチップではプロセッサ間の相互接続網の動作周波数を高くすることができるため、プロセッサ間での高速の通信と低オーバーヘッドの同期が可能になると思われる。このようなハードウェアの進歩によりマルチプロセッサシステムでの細粒度並列性の利用が現実的になると期待される。

従来、細粒度並列性を利用した並列処理ではレスポンスタイムを最小化することに主眼が置かれていた。これに対し、汎用計算機としてはレスポンスタイムとスループットの両立が重要であるので、プロセッサ利用率を向上させる静的／動的スケジューリングのハイブリッドなスケジューリング手法の開発が必要である。

細粒度並列性を利用する場合、プロセッサ間で頻繁に同期・通信を行う必要がある。このような実行形態に適した同期機構としてバリア同期機構がある。Fuzzy Barrier[7] や Elastic Barrier(一般化されたバリア型同期機構)[3]、Ultimate Barrier(重複可能なバリア型同期機構)[4] を利用すれば、本来同期の必要でないプロセッサに挿入する同期によるオーバーヘッドをわずかに抑えることができるため、ハードウェアによるバリア同期順序制御の機能を持たないバリア同期機構が使用できる。このようなバリア型同期機構では、バリア同期を取ろうとするプロセッサ群を同期を行う前にあらかじめ設定しておく。同期をとるプロセッサ群は再び設定しなおすまで以前の状態を保持する。同期成立毎にプロセッサ群の組合せが変化する Barrier Queue[8] や Barrier Dag Memory[9] などの機構を持たないため同期検出のハードウェアを単純にできる。

そこでこのようなバリア同期機構を想定し、同期を取るプロセッサ群の設定をスケジューラがプロセッサの割り当てと同時に行う方法を提案する。応用プログラムは複数の並列コード群に分割され、実行可能状態になると動的にスケジューリングされる。これによりプログラムの並列性の変動および非決定性によって生じるプロセッサ利用率の低下を軽減する。

本稿は以下のように構成されている。第2章では本稿が想定するアーキテクチャを規定する。第3章でプログラムの実行モデルとスケジューリング方法を述べる。第4章でタスクフレームでどのようにループを表すかを示す。第5章で本スケジューリング法の評価をおこなう。最後に第6章で簡単にまとめを述べる。

2 前提

2.1 対象アーキテクチャ

図1に本稿が対象とするアーキテクチャを示す。

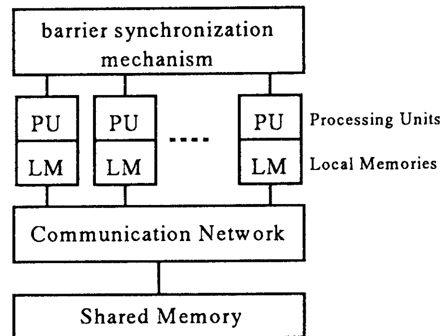

図1　対称とするアーキテクチャ

相互接続網を通して全てのプロセッサからアクセスできる共有メモリを持ち、各プロセッサ毎にローカルメモリを持つ対称型の密結合マルチプロセッサシステムとする。

共有メモリへの不可分なアクセスを保証するために相互排除機構を持つ。また、高速の同期機構として、次節で述べるようなバリア同期機構を持つものとする。

2.2 バリア同期機構

バリア同期機構を3つの側面から規定する。

まず第1は同期検出の多重化の有無であり、同時に複数のバリア同期を処理できるかを示す。同期検

出の多重化は associative memory と OR buffer などによって実現できる [5][8][9]。

第2はバリアグループの設定もしくは指定方法である。ここで、バリアグループとは同期を取り合うプロセッサ群である。この方法としてはタグ方式：同期命令を発行する毎に同時にバリアグループを指定する [1][2]、予約方式：バリアグループをキュー構造などでまとめて予め登録しておき同期が成立する毎に自動的に更新する [8][9]、変更命令方式：バリアグループの変更が必要になる毎に変更命令を発行する [5] 等が提案されている。ハードウェア量はタグ方式 > 予約方式 > 変更命令方式 の関係にあるが、変更命令方式では実行時にソフトウェアによるバリアグループの管理が必要になる。

第3はバリアの概念の拡張であり Fuzzy Barrier、Elastic Barrier、Ultimate Barrier などのバリア型同期が提案されている。バリア型同期は同期の成立条件を緩やかにすることにより同期待ちにより発生するアイドル状態を減少させ、また本来同期の必要でないプロセッサに挿入する同期によるオーバーヘッドを削減することができる。Ultimate Barrier を用いた場合を例にとると、コード間の依存関係が実行前にあらかじめわかっている場合には問題が要求していない命令間に同期機構に依存した先行制約を付加することによる性能劣化は数%で済むことが実験的に確かめられている [4]。

本稿では多重化された変更命令方式のバリア同期機構で任意のバリア型同期をサポートするものを想定する。この方式ではバリアグループの変更頻度とバリア同期によるオーバーヘッド、細粒度並列化の度合、そしてプロセッサの割り当ての効率がシステム性能に大きな影響を及ぼす。そこで静的スケジューリングにより細粒度並列性を利用し、粗粒度並列性に基づいて動的にプロセッサ割り当ておよびバリアグループ設定を行うことで効率化を図る。

3 スケジューリング

3.1 実行モデル

実行モデルを図2に示す。プロセスはOSによる記憶空間などの資源割り当ての単位である。プロセスは複数のタスクフレームから成る。タスクフレームはプロセッサ割り当ての単位であり、細粒度に並列化された命令ストリーム群とその間で共有される変数、および他のタスクフレームとのインターフェースの集合体である。図3にタスクフレームの構造を示す。タスクフレームは複数のインレットと1つのスレッドをもつ。インレットはタスクフレーム内のローカル変数へ他のタスクフレームで生成されたデータを受けいれるために外部に公開されたプロシジャである。 インレットは受け取ったデー

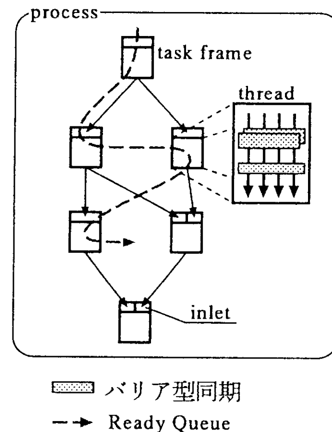

図2　実行モデル

タをタスクフレーム内のローカル変数領域に格納する。インレットは必要なデータが揃ったことを同期カウンタによって検出するとタスクフレームをスケジューラのレディーキューに登録する。

スレッドは複数のプロセッサにより並列実行される並列化された命令ストリーム群の集合であり、タスクフレーム内にスレッドの実行に必要なプロセッサ数が格納されている。スケジューラは必要な数のプロセッサを確保しタスクフレーム内のスレッドポインタが指し示すアドレスのコードを各プロセッサに実行させる。スケジューラはまた、それらのプロセッサを1つのバリアグループとしてバリア同期機構に登録する。各プロセッサがスレッド内の対応する命令ストリームに分岐するように、スケジューラは各プロセッサに一意な識別子を与える。

スレッドがデータの生産者である場合、消費者のタスクフレーム内のローカル変数に対して消費者の

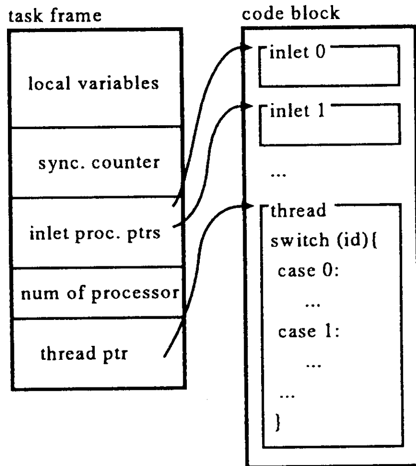

図3　タスクフレーム

タスクフレームが公開しているインレットを通してデータを送る。タスクフレーム間に制御依存がある場合には、インレットを通して空のデータを送ることにより依存先のタスクフレームをレディーキューに登録させる。

スレッドを実行中のプロセッサの状態を図4に示す。 コードブロックは共有メモリへのアクセスの

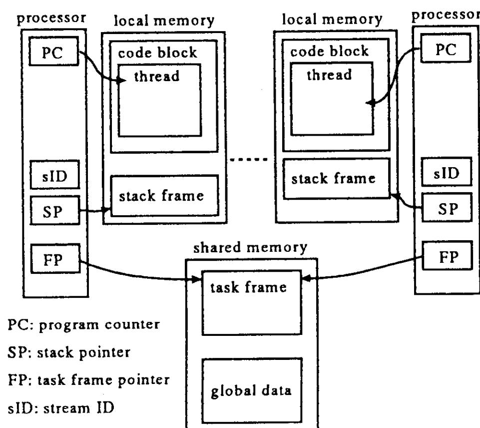

図4　実行時のプロセッサの状態

コストが高い場合にはローカルメモリに、そうでない場合には共有メモリに置かれる。どちらの場合でも、コードブロックは全てのプロセッサから同一アドレス上に見える。

データ領域には共有メモリ内のグローバル領域、タスクフレーム、そしてローカルメモリ内のスタックの3ケ所がある。各データ領域にはそれぞれプロセス全体で共有するデータ、スレッド内の命令ストリーム間で共有するデータ、そして命令ストリーム内のローカルなデータを格納する。

3.2　タスクフレーム内スケジューリング

原始プログラムをループや条件分岐、または依存グラフ内での依存関係の密な部分グラフなどから複数のタスクフレームに分割する。粗粒度の並列性はタスクフレーム間で表現する。

スレッドはリストスケジューリング等を用いてプロセッサ間の通信／同期などのコストを考慮しながら細粒度の並列性を利用して静的にスケジューリングする。命令ストリーム間の実行順序を保証するために必要な位置にバリア命令を挿入するが、スレッド内の各命令ストリームは常に同一のバリアグループに属するプロセッサによって実行されるため本来同期の必要でない命令ストリームにもバリア命令をいれる。同期による待ち時間の発生をできるだけ抑えるために、適切なバリア型同期を使用する。

スレッドのスケジューリングに際しては不必要に多くのプロセッサを利用しないように留意する。並列化による速度向上率とプロセッサの利用率から適切なプロセッサ数を決定する。

3.3　実行時スケジューリング

タスクフレームのスケジューリングは実行時に動的に行う。実行に必要なデータの揃ったタスクフレームはそのインレットによりスケジューラのレディーキューに登録される。

ハードウェアバリア同期は実際に協調するプロセッサが同時に動作することを想定した同期機構であるため同期が頻繁に行われ、また同期と同期の間隔が比較的狭い。そのため必要な数のプロセッサをほぼ同時に割り当てなければならない。スケジューラはレディーキューからプロセッサ割当を行うタスクフレームを選択する。

バリアグループの登録と各プロセッサのスレッド実行開始のタイミングはどのバリアグループにも属していないプロセッサが同期要求を出した時の同期機構の振舞により若干異なる。

同期不成立と解釈する場合：プロセッサは後にいずれかのバリアグループに含められ、同期が成立するまで停止する。そこでスケジューラは利用可能なプロセッサを得たらすぐにどのバリアグループにも属さないようにしスレッドの実行を開始させる。最後に割り付けを行われるプロセッサは、それまでに割り付けられたプロセッサ群を含めて1つのバリアグループとして同期機構に登録し、その後スレッドの実行を開始する。

その他 (同期成立と解釈するかエラーとなる)：必要な数のプロセッサが利用可能になったところで、それらのプロセッサを1つのバリアグループとしてバリア同期機構に登録する。その後、各プロセッサはスレッドの実行を同時に開始する。

前者のスケジューリングの方が、若干オーバーヘッドを縮小できるのでできるだけ前者を採用する。

スレッドの実行が終った後の処理は Elastic Barrier に見られるダミーの同期要求をサポートするかしないかによって異なる。

ダミーの同期要求をサポートしない場合：スレッドの処理を終了したプロセッサが1つでもある場合には、そのスレッド内でバリア型同期は全て完了している。従ってスレッドの処理を終った最初のプロセッサが、属していたバリアグループをすぐに消去する。

ダミーの同期要求をサポートする場合：図5に示すように、最後にダミーの同期要求を発行したプロセッサがスレッドの処理を終了し、スケジューラのバリア変更コードまで実行が進んだとしても、他のプロセッサはまだ全てのバリア同期を完了していない場合がありえる。 そのため、スレッドの実行を完了したプロセッサは自分を除いたものにバリアグループを登録しなおすか、全ての同期が完了するのを待たなければならない。もしくは、スレッド内で最後に実行する同期にはダミーの同期要求を使用しない。

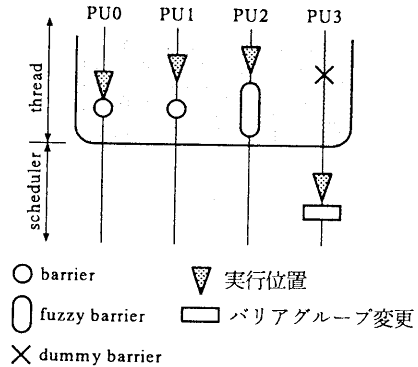

図5　ダミーの同期要求をサポートする場合

4　タスクフレームによるループの表現

4.1　DO-ACROSS ループ

DO-ACROSS ループはループのイタレーション間に依存があるため、その並列性はコンパイル時に決定できることが多い。

図6に依存グラフで表現した DO-ACROSS ループの例を示す。 このループはソフトウェアパイプ

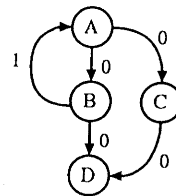

図6　DO-ACROSS ループの例

ライニングを用いて並列化され図7に示すタスクフレームを形成する。

4.2　DO-ALL ループ

DO-ALL ループでは各イタレーションを並列実行する。DO-ALL ループの効率的な動的スケジューリング法として Guided Self-Scheduling(GSS) がある [10]。タスクフレームでは次のようにして GSS を利用することができる。

例としてイタレーション数25、全プロセッサ数

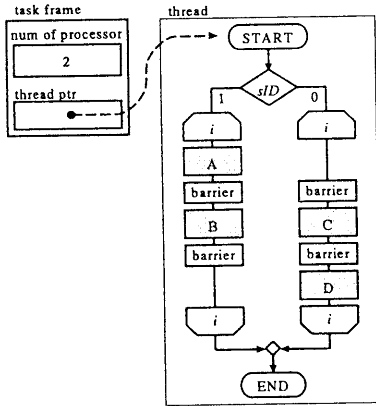

図 7　DO-ACROSS ループのタスクフレーム

5 台の場合を考える。GSS(2) によれば初めに割り付けられたプロセッサから順に 5, 4, 4, 3, 2, 2 だけの数のイタレーションを実行する。これを表すために 6 個のタスクフレームを生成し、図 8に示すようにフレーム内ローカル変数を初期化する。*begin_i* と *end_i* がそれぞれ各タスクフレームが実行を開始するイタレーションと終了するイタレーションを示す。 フレーム 0 から順にインレットを通してロー

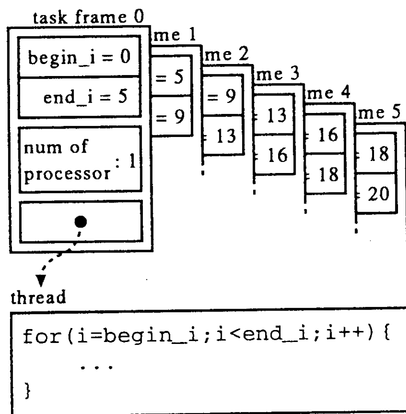

図 8　DO-ALL ループのタスクフレーム

カル変数を設定するとそのフレームはスケジューラのレディーキューに登録され、それぞれ 1 台のプロセッサにより実行される。

5　評価

5.1　実験システム

本スケジューリング法をマルチプロセッサシステム MSBM 上で評価した。MSBM の概要を図 9に示す。

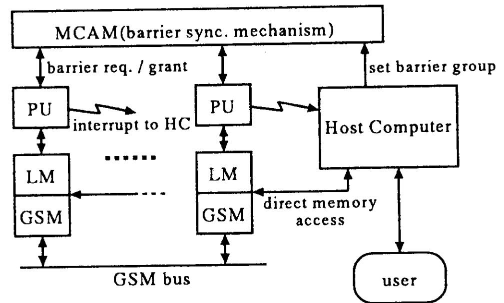

図 9　MSBM の概要

MSBM は対称型マルチプロセッサシステムであり、ローカルメモリ (LM) および部分分散共有メモリ (GSM) を持つ。プロセッサ群を複数のグループに分け、ハードウェアによりそれぞれのグループで個別に GSM のコヒーレンシを保つことができる。

バリア同期機構は包含検索機能をもつ MCAM による集中型機構であり、複数の同期成立を同時に検出できる多重化バリア機構である。バリア同期として通常のバリア同期と fuzzy barrier を行うことができる。同期要求はバリア命令によって行う。fuzzy barrier は先行命令とバリア命令により同期領域をはさむことによって表現する。Ultimate Barrier のような重複の機能は無い。バリアグループの変更はホストコンピュータから行う。

相互排除として同一グループ内の他のプロセッサに対するロック命令をもつ。ロックをかけたプロセッサがアンロック命令を実行するまで、同一グループの他のプロセッサは動作を停止する。

プロセッサは全て同一グループに設定し、全 PU の GSM を同一内容とする。評価プログラムは全てのプロセッサのローカルメモリにあらかじめロードしておく。スケジューラはシステム全体で 1 つのレディーキューを持ち、並行動作する各プロセスが

登録するタスクフレームはレディーキューに混在する。

5.2 ベンチマーク

Whetstone ベンチマークプログラムを本システム向けに並列化し、複数プロセスの同時実行による平均プロセッサ利用率とレスポンスタイムの変動を測定した。それぞれ図10と図11に実験結果を示す。

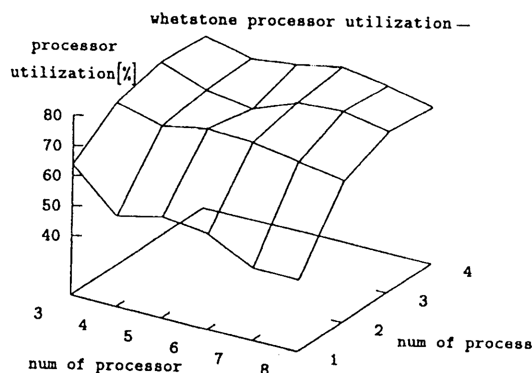

図10 平均プロセッサ利用率

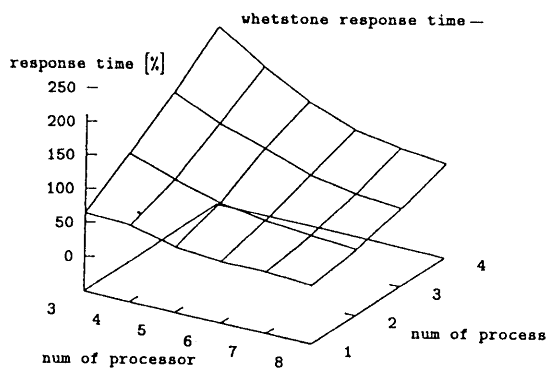

図11 レスポンスタイム

各図において、並列実行数は同時に実行するプロセスの数を表している。プロセッサ利用率はプロセッサの全実行時間に対するプロセスのコードの実行に要した時間の割合とし、レスポンスタイムは並列化を行っていない逐次版のプログラムの実行時間に対する比で表している。

図10より、プロセッサ数が増加ほど、平均のプロセッサ利用率は低下する傾向にあること、プロセスを複数並行に実行することでプロセッサ利用率が向上していることがわかる。並列実行数を2以上にしてもそれほどプロセッサ利用率が向上していないことから、並列実行数2ですでにスケジューラのレディーキューに常に割り付け待ちのタスクフレームが存在するようになっていることが予想される。

図11より、レスポンスタイムはプロセッサ数が少なく、プロセスの並列実行数が多いほど長くなることがわかる。測定を行った領域のうちほぼ半分の領域で並列実行を行わないものよりもレスポンスタイムが長くなっている。これは細粒度並列処理を行ったことによるプロセッサ利用率の低下と、動的スケジューリングによるオーバーヘッドによると思われる。

6 むすび

ハードウェアが単純な変更命令方式のバリア検出機構を想定し、プロセッサのスケジューラがバリアグループを管理する方法を提案した。Whetstone ベンチマークプログラムを本システム向けに並列化し評価し、複数のプロセスの同時実行によりプロセッサ利用率が向上することを示した。システムの負荷が上昇するに従いレスポンスタイムは増加し、負荷があるレベルを越えると逐次版のプログラムよりも長くなることがわかった。

今後の課題として、DO-ALL ループでの性能評価およびより効率的な表現法の考案、システムの負荷に応じた並列性の調整、プロセッサのコンテキストスイッチングおよびキャッシュを考慮した場合のスケジューリング法の考案などが挙げられる。

謝辞

本研究に支援していただいている (株) 東芝日野工場 木村勝彦氏に深謝致します。

参考文献

1) 高木 浩光, 河村 忠明, 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の構成法, 並列処理シンポジウム JSPP'90 論文集, pp. 57-64(1990).

2) 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実行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, pp. 269-276(1991).

3) 松本 尚, Elastic Barrier: 一般化されたバリア型同期機構, 情報処理学会論文誌, Vol. 32, No. 7, pp. 886-896 (1991).

4) 高木 浩光, 有田 隆也, 曽和 将容, 重複可能なバリア型同期のためのスケジューリングアルゴリズムとその性能, 電子情報通信学会研究会資料 CPSY91 - 15, pp. 91-97(1991).

5) 本石 彰, 濱田 智雄, 岩根 雅彦, 米沢 敏夫, 細粒度マルチマイクロプロセッサ MSBM の構成と性能評価, 情報処理学会研究報告, 94-ARC-107, pp. 1-8 (1994).

6) 岩根 雅彦, 野口 善昭, 茶屋道 宏貴, 本石 彰, 重松 保弘, 細粒度並列計算機 MSBM の開発, 九州工業大学研究報告 (工学), No. 67, pp. 61-68(1995)

7) R.Gupta, The Fuzzy Barrier: A Mechanism for High Speed Synchronization of Processors, *Proc. Third Int. Conf. ASPLOS, pp. 54-63(1989).*

8) M.T.O'Keefe and H.G.Dietz, Hardware Barrier Synchronization: Static Barrier MIMD (SBM),*1990 Int'l Conf. on Parallel Processing vol.I, pp. 35-42(1990).*

9) M.T.O'Keefe and H.G.Dietz, Hardware Barrier Synchronization: Dynamic Barrier MIMD (DBM),*1990 Int'l Conf. on Parallel Processing vol.I, pp. 35-42(1990).*

10) Constantine D. Polychronopoulos and David J. Kuck, Guided Self-Scheduling: A Practical Scheduling Scheme for Parallel Supercomputers, *IEEE Trans. Compt., vol. C-36 no. 12 pp. 1425-1439(1987)*