マルチスレッド計算機における同期機構とパイプライン構成

坂井修一[†,☆] 岡本一晃[††] 松岡浩司[††]
廣野英雄[††] 横田隆史[††]

本論文では，マルチスレッド型超並列計算機における2つの基本機能である同期とパイプラインについて検討し，これらの効率的な実現方式の提案を行う．最初に，これまでに提案されたデータ駆動型の同期機構について，効率とハードウェアの複雑さの観点から検討を行い，同期機構の最適化について提案する．次に，この同期機構を含むパイプラインの構成法について述べる．さらに，提案の方式について基礎的評価を行い，コストパフォーマンスに関する考察を加え，関連事項をあわせた検討を行う．最後に将来の課題に関して述べる．

Synchronization and Pipelining on a Multithreaded Computer

Shuichi Sakai,[†,☆] Kazuaki Okamoto,[††] Hiroshi Matsuoka,[††]
Hideo Hirono[††] and Takashi Yokota[††]

This paper examines two basic functions in a multithreaded massively parallel computer — synchronizations and pipelining — and proposes efficient implementations of them. The data driven synchronization mechanisms which have been proposed are carefully analyzed from the viewpoint of efficiency and hardware complexity, and the optimized synchronization mechanism is proposed. The pipeline structure for a massively parallel computer containing the proposed synchronization is also presented. Performance improvement methods for this pipeline are proposed. In addition, basic evaluations of the proposed mechanisms are carried out, the cost-effectiveness of the proposed method is considered, and related issues are examined. Lastly, future problems are presented.

1. はじめに

データ駆動アーキテクチャの研究は，1980年代末より純粋なデータ駆動計算機からマルチスレッド計算機にその対象が移行してきている．その代表的なものは，(1) マサチューセッツ工科大学における TTDA[2] から Monsoon[11] や Empire[5] へ，さらには Monsoon から *T[13),15),24)] への流れ，(2) 電子技術総合研究所における SIGMA-1[4] から EM-4[7] へ，さらには EM-4 から EM-X[23] への流れ，(3) サンディア国立研究所における Epsilon-1 から Epsilon-2 への流れ[10] である．

データ駆動アーキテクチャの発展形としての細粒度マルチスレッドアーキテクチャは，(1) 大きなレーテンシをスレッド切替えによって隠蔽する点，(2) メッセージ処理と同期の高速化によって効率的な細粒度並列処理を行う点，(3) 並列性を自然かつ最大限に抽出できるプログラミングモデルとの相性がよい点，などにおいて，データ駆動アーキテクチャの利点を最大限に活かすアーキテクチャである．さらに，このマルチスレッドアーキテクチャは，内部にフォンノイマンアーキテクチャを含んでおり，(4) 直列ないし局所並列型の計算に適している点，で，より実用的なアーキテクチャとなっている．またフォンノイマンアーキテクチャを RISC 化し，さらにスーパスカラ化することで，より高効率で局所並列性を活かすアーキテクチャを実現することが可能である．

一方マルチスレッドアーキテクチャは，データ駆動

† 電子技術総合研究所情報アーキテクチャ部
Information Science Division, Electrotechnical Laboratory
†† 新情報処理開発機構
Real World Computing Partnership
☆ 現在，筑波大学電子・情報工学系
Presently with Institute of Information Sciences and Electronics, University of Tsukuba

アーキテクチャからの拡張だけに限られるわけではない．たとえば，Denelcor 社の HEP[1]とその後継機である Tera Computing System 社の MTA[6]は 1 つのパイプラインが複数のスレッドの実行を重畳化して行う典型的なマルチスレッド計算機である．Dally の J-Machine[16]は複数のスレッドをインタリーブするわけではないが，スレッド切替えを高速化していることから，マルチスレッド計算を支援するアーキテクチャと見なすことができる．Dally のグループでは，J-Machine の後継機である M-Machine を開発中であるが，これは多くのスレッドがプロセッサチップ内でパイプラインを共有する機構を持っており[17]，マルチスレッド計算機である．

これらの計算機の開発において，共通する重要課題として，(1) 通信の高速化，(2) 同期処理の高速化，(3) コンテクスト切替えの高速化，などがあり，さらに，これらの機能を統合しかつ潤滑に処理を進めるようなパイプラインのデザインが，マルチスレッド計算機の効率に決定的な影響を及ぼすといえる．

本論文は，特にマルチスレッド計算機の同期機構と，それを活かすパイプラインの機能について検討し，より高度化した機構の提案を行う．最初に，これまでに提案されたデータ駆動型の同期機構について効率とハードウェア量の両面から再検討し，新しい同期機構の提案を行う（2 章）．次に，この同期機構を有効に活用する超並列計算機の要素プロセッサ（PE）のパイプライン設計について検討する（3 章）．続いて，このパイプラインを有する計算機を想定して，その基礎的評価を行う（4 章）．さらに，提案した同期方式・パイプライン方式のコストパフォーマンスに関して考察し，関連事項の検討を行う（5 章）．最後に，将来の課題について述べる（6 章）．

本論文で述べる機構は，現在の VLSI 技術を用いて十分に実装可能である．これにより，通信・同期の性能が計算性能と同程度（厳密に等しいスピードか，悪くても数分の一以上の性能）となり，各プロセッサがメッセージを入力してからメッセージを出力するまでの遅延が最短で 4 クロック（RISC クロック）であるような超細粒度の超並列計算機を開発することが可能となる．

本論文は，新情報処理開発機構（RWCP）にて開発中の超並列計算機 RWC-1[22]のプロセッサ RICA-1 の基礎となるアーキテクチャを提案したものである．RICA-1 は，本論文の内容に加えて，並列 OS の支援，入出力機能の強化といった重要な要素技術を持つ．

2. 同期機構の高速化

2.1 データ駆動型同期機構

マルチスレッド計算機の同期機構を検討する前に，まずその基礎となるデータ駆動型同期について考える．ここではすでに提案されているデータ駆動型の同期機構を取り上げ，それらの利点および欠点について検討する☆．

初期の動的データ駆動計算機は，データ駆動型の同期（トークンマッチング）のためにハッシュなどの連想機構を持っていた．たとえば，マンチェスタ大学のデータ駆動計算機[3]は，トークンマッチングのための疑似連想アクセスを並列ハッシュテーブルを用いて実現していた．また，SIGMA-1[4]においては，PE に連鎖ハッシュ機構が組み込まれていた．ハッシュによるマッチングは，次のような利点がある．

- 高いメモリ使用効率
- 実現するマッチング方式の柔軟性：一例としてスティッキー・トークンの実現

他方で，以下のような欠点が指摘されていた．

- ハッシングに要するハードウェア量およびハードウェアの複雑さ
- ハッシュミスの場合のパイプラインバブルまたは例外処理
- トレースの複雑さ

その後のデータ駆動アーキテクチャの研究において，これらの欠点はセグメントを基本とする同期機構の導入によって解決された．Monsoon[11]における ETS（Explicit Token Store），EM-4[7]における直接マッチング，Epsilon[10]における direct match などがその機構である．

これらの同期機構においては，マッチングに用いるメモリの領域が，排他的に 1 つの関数インスタンスに割り当てられる．この領域のことをオペランドセグメントと呼び，関数インスタンスのコード領域をテンプレートセグメントと呼ぶ．オペランドセグメントは，関数起動のときにメモリの空き領域中に割り付けられる．データ待合せは，当該オペランドセグメントの特定の語の上で行われる．この待合せ場所の番地は，オペランドセグメント番号（OSN, Operand Segment Number）とオペランドセグメント上での変位（ODPL, Operand DisPLacement）との組で表される．そして同期と，それに引き続いて起動される関数

☆ ここでは，著者は動的データ駆動モデルに基づく同期のみを対象とし，静的データ駆動モデルに基づくものは，対象としない．

の命令列との対応づけは，待合せ番地 (OSN, ODPL) と命令番地との対応をとることで，実現される．命令番地は，テンプレートセグメントの番号（TSN, Template Segment Number）とテンプレートセグメントの変位（TDPL, Template DisPLacemnet）との組で表される．

また，関数実行の最後に，オペランドセグメントは将来の使用に備えてリリースされる．

これらの同期機構の利点は以下のとおりである．

- 連想機構が不要である．
- ハッシュミスによるパイプラインの擾乱がない．
- 待合せの番地が命令に書かれているため，ハッシュを用いた場合と比較して，デバグやトレースが容易である．

一方，Monsoon と EM-4 では関数を起動する際，待合せの番地と命令番地との対応づけの実現法に違いがあり，それぞれに特有の欠点を有している．

まず Monsoon が採用する ETS では，命令によって同期機構を起動する．トークンには OSN と命令番地 (TSN, TDPL) を持たせており，ODPL は命令の一部として実装される．したがって，

- マッチングの成否にかかわらず，必ず命令フェッチが必要になる．
- 命令フェッチとマッチングが直列化されなければならず，前者は必ず後者より先に実行される．これによって，パイプライン長が長くなり，ターンアラウンド時間が長くなる．

といった欠点がある．

これに対し EM-4 が採用する直接マッチング法では，命令ではなくトークンの到着が自動的に同期機構を起動する．トークンには待合せ番地 (OSN, ODPL) を持たせており，オペランドセグメントの先頭に TSN を格納することで，同期と起動する命令列との対応を実現している．ここで，TPDL は，OPDL と兼用することで実現している．このため直接マッチング法では，

- 命令フェッチに先行して TSN を獲得する必要があるので，ターンアラウンドが 1 サイクル長くなる．
- 関数起動のときに，TSN をオペランドセグメントの最初に書き込む必要がある．関数本体が小さいときに，このオーバヘッドが問題となる．
- すべての命令が，オペランドセグメントの 1 語を必ず消費することになるので，メモリの使用効率を落とす．

といった欠点がある．

なお，Monsoon でも EM-4 でも，2 入力マッチングを 1 クロックで実行する．すなわち，メモリへのリードモディファイライトは 1 クロックで排他的に実現され，パイプラインのピッチはこれより小さくはならない．

表 1 データ駆動型同期機構の比較
Table 1 Comparisons of data driven synchronizaion mechanisms.

	Monsoon	EM-4	*T
起動	命令	トークン	命令
待合せ番地			
OSN	トークン	トークン	トークン
ODPL	命令	トークン	命令
命令番地			
TSN	トークン	オペランドセグメント先頭	トークン
TDPL	–	トークン（共用）	–
パイプライン長	4	2	5 以上
メモリ使用効率	○	×	○
マルチウェイ・ジョイン	×	×	○

その後，Monsoon の後継機として開発された *T [13),15),24)] では，データ駆動型の同期は ETS に基づくが，Monsoon の場合より柔軟性が高い．スレッド処理に基づく計算のためには，3 個以上のメッセージの同期をとる機構が必要である．一般に n 個のメッセージの同期を効率的かつ柔軟に行うために，*T は各スレッドの先頭で不可分の命令列を組み，汎用の Join を実現している．*T におけるメッセージは，Monsoon の ETS と同様に OSN と (TSN, TDPL) を持っている．Join は，フラグ（またはカウンタ）を読み出し，これをインクリメントし，同期が完了したかどうかを見て，条件分岐する，というプロセスである．この方式は，ターンアラウンド時間が長くなるが，柔軟性が高い．

主なデータ駆動型同期機構について，その比較を**表 1** に示す．

2.2 マルチスレッド計算機の同期機構

データ駆動計算機の場合と異なり，マルチスレッド型の同期で必要となるのは，次のようなことである．なお，ここで，「スレッド」を 1 プロセッサで連続して実行する命令の列，と定義する．その大きさは任意であるが，本論文では，数命令から 10 命令程度以上のスレッドを対象と考える．

(1) マルチスレッド型アーキテクチャにおいては，3 個以上の入力トークンをともなうマルチウェイ・ジョインが，頻繁に起こる．これは，一般に長いスレッドの実行は，多くのデータを必要とする場合が多いからである．これに比較して，データ駆動計算

機においては，スレッドは1命令からなるため，必要とされるデータ量も少ない．

さらに，トークンの表現形式であるメッセージについても，固定長でないほうが効率的である．というのも，メッセージの持つデータ量は，プログラムのデータ分配方式や引数渡しの方式に依存して1個から多数までさまざまだからである．

(2) マルチスレッドアーキテクチャにおいては，従来のデータ駆動計算機ほど頻繁に同期が起こらない．これは，スレッドの内部では，マッチングが不要だからである．

(3) 局所性を利用することが効率上重要であり，キャッシュやレジスタセットがスレッドの実行のために利用されなくてはならない．同期は，これら局所的な記憶装置の処理速度と同じぐらいの速度で実現されるのが望ましい．

1番目の要求によって，従来の2入力マッチングの機構に加えて（の代わりに）カウンタ型の同期機構が必要となる．さらに，複数の待合せデータが待合せメモリ上に記憶されるような機構を考えなければならない．2番目の要求は，同期のためにどこまでのコストをかけるべきか，という問題にかかわってくる．極端な場合，もし同期が百万クロックに1度程度しか起こらないのであれば，同期のために特殊な機構を用意する必要はない．3番目の要求によって，同期は短いピッチのパイプラインで実現することが重要となる．さらに，命令フェッチやデコードとマッチングの並列化も考えるのが望ましい．

以上の要件を踏まえ，マルチスレッド計算機のための同期機構を提案する．

2.3 高速化・高機能化された同期機構

ここでは，マルチスレッド計算機のためにより高速化・高機能化された同期方式を提案する．その基本性能については4章で，コスト性能比については5章で考察することにする．

データ駆動型の同期をさらに高速化するために，ETSや直接マッチング法が持つ利点を継承し，かつそれぞれの欠点となるオーバヘッドを解決する同期機構を以下に提案する．

図1 に並列マルチスレッド計算機において高機能化された同期機構を示す．

本方式は直接マッチングを基本とする．各メッセージは (TSN, TDPL) のペア（= 命令番地）と (OSN, ODPL) のペア（= マッチングの番地）を含んでいる☆．したがって，変位のために2つのフィールドを持ち，命令番地とマッチング番地は完全に独立したものとなる．

さらに，メッセージには同期フラグフィールド（SF, Synchronization Field）があり，同期の型を示している．SFは，改良型直接マッチングより大きく，3ビット以上のフィールド長を持つ．SFは，本メッセージが，(1) 同期なし（monadic），(2) 2入力左（dyadic left），(3) 2入力右（dyadic right），(4) 3入力以上の同期の入力（more than two messages），(5) シグナル同期（signal），(6) 2入力左で相手が即値のもの（dyadic left with immediate），(7) 2入力右で相手が即値のもの（dyadic righ with immediate），のどれであるかを示す．スティッキートークンを表現したり，3入力同期用に特殊な型を設けたりする場合には，さらに大きなフィールドが必要になる．

関数起動時には，オペランドセグメントが確保されるが，TSNのオペランドセグメント先頭番地への格納は不要である．トークンがプロセッサに到着すると，TSNとTDPLを用いて，即座に命令フェッチが始められる．同時にOSNとODPLを用いてマッチングが行われる．

命令キャッシュとデータキャッシュを，局所メモリのレーテンシ短縮のために用いる．これらは，通常のマイクロプロセッサで用いられているように，オンチップの一次キャッシュとして実現され，必要に応じて二次キャッシュを追加する．さらにここでは，独立したキャッシュであるフラグキャッシュを設けて，同期処理の独立性を高め，高速化をはかる．

ここでは，一次キャッシュを読む（あるいは書く）時間を1クロックとする．メッセージがプロセッサの同期機構に入ってくると，即座に命令がフェッチされる．命令フェッチとデコードは，同期と完全に独立・並列に行われ，デコードの終了と同期の成功が確定したところで命令実行が行われる．

実際には，SFの中身によって5種類のデータ駆動型同期がある．以下に各場合について記す．

(1) 1入力の場合（Monadic）

スレッドの最初の命令が引数データを1つしかとらない命令であったり，この命令を発火させるメッセージが1つですべての引数データを運んできた場合，同期は不要である．

当該命令のフェッチ，デコードのほかに，命令実行以前には何も起こらない．この場合，ターンアラウンド短縮のために，パイプラインの同期ステージはバイパスされなければならない．

☆ 一般に，すべての番地は仮想化されており，各プロセッサにはアドレス変換機構が入っている．

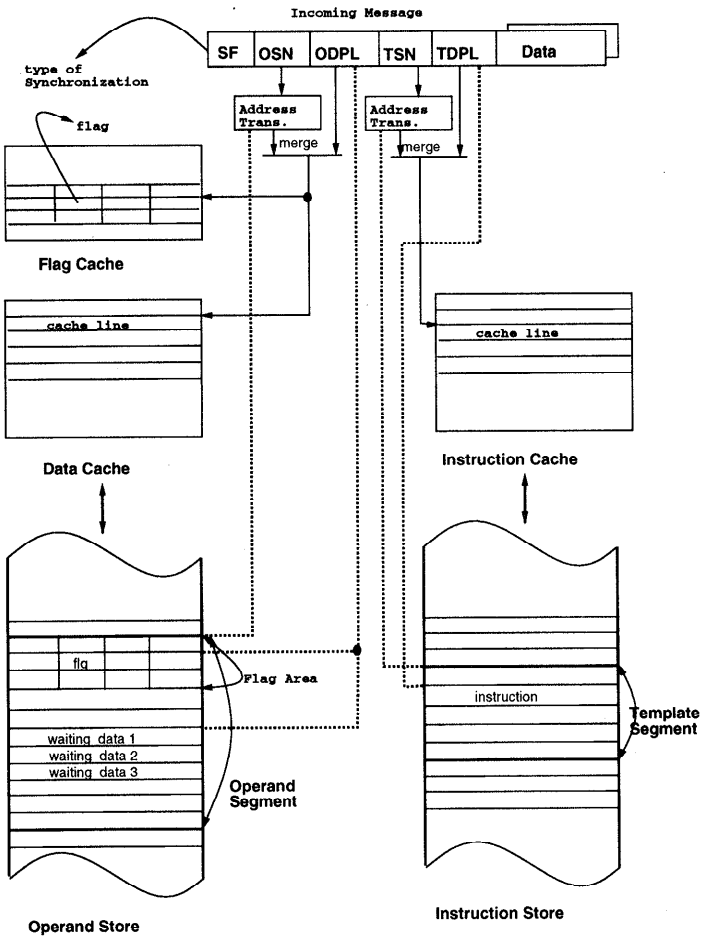


図1 高速化・高機能化された同期機構
Fig. 1 Improved synchronization mechanism.

(2) 2入力の場合（Dyadic）

2入力の同期は，2入力マッチング方式をとる．これは(3)に示すマルチウェイ・ジョインによっても実現することができるが，2入力の場合だけは専用ハードウェアを付加することにする．2入力の同期は頻繁に起こるものであり，その性能が並列計算機の全体性能に大きな影響を及ぼすからである．

2入力の同期は，1) フラグキャッシュからフラグを読み込む（キャッシュミスを起こした場合は，外部メモリからキャッシュラインを読み込みそこからフラグを読み込む），2) フラグを見て，相手のデータが到着しているか否かを調べる，3) 相手がいればフラグをクリアし，相手がいなければフラグをセットする，4) データキャッシュの (OSN, ODPL) の番地からデータを読み込む（キャッシュミスを起こした場合には，外部メモリからキャッシュラインを読み込み，そこからデータを読み込む），5) 同期の相手が不在の場合には，(OSN, ODPL) の番地にデータを書き込む，という手順で行われる．ここで，1), 2), 3) の操作は不可分である．

4) の操作は本来，2) が終了し，相手の存在が確定してから行われるものであるが，副作用なく投機的に実行することもできる．その場合，キャッシュミスが起これば，相手の存在が確認されるまで待つのが効率的である．

上記の手順は並列化される．具体的には，1) と 4)（投機実行）が並列化され，さらに 3) と 5) が並列化される．したがって，同期の手順は，次のようになる．

クロック1：1) と 4) の同時実行

クロック2：2) に引き続いての 3) と 5) の同時実行（相手がいた場合は 5) は実行されない）．

2) は単純な操作であり，1) のときから番地は保持されているので，2) と 3) は1クロックのうちに処

理可能である．

完全なパイプライン化のためには，2ポートのフラグキャッシュと2ポートのデータキャッシュが必要である．

(3) マルチウェイ・ジョイン（≥ 3）

3入力以上の同期を実現するには，以下の2つの方法がある．第一の方法は，2入力同期を木状（あるいはカスケード状）に配するやりかたである．木の各ノードは2入力マッチングを行い，メッセージのデータをメモリ上の特定番地に書き込む．この方法は，上に述べた「2入力の場合」以外のハードウェアを必要としない．しかし，これは中間ノードのために多くの命令と多くのメッセージが必要となる方法である．より正確には，n 入力のジョインには，$n-1$ 個の命令が必要であり，$n-2$ 個のメッセージが新たに生成されることになる．

マルチウェイ・ジョインをより効率的に，またより柔軟に実現するためには，ジョインカウンタの計数回路が必要となる．この場合，フラグは到着したメッセージの数を数えるカウンタとして用いられる．カウンタの値がスレッド実行に必要な入力メッセージの数 n になったとき，スレッドは発火する．

マルチウェイ・ジョインは，1) フラグキャッシュからフラグを読み込む（キャッシュミスを起こした場合は，外部メモリからキャッシュラインを読み込み，そこからフラグを読む込む），2) フラグ（＝カウンタ）の値をインクリメントし，結果を n と比較する，3) 2) が等しければフラグをクリアし，2) が等しくなければフラグの値をキャッシュに書き込む，4)（ジョインに成功した場合）待合せ中のデータをデータキャッシュから読み込む（キャッシュミスを起こした場合は，外部メモリからキャッシュラインを読み込み，そこからデータを読み込む），5)（ジョインに失敗した場合）メッセージのデータをデータキャッシュに書き込む（キャッシュミスを起こした場合は，外部メモリからキャッシュラインを読み込みここにデータを書き込む），という手順で行われる．1)，2)，3) の操作は不可分である．

ここで，4) は特殊な回路を設けなくても，通常のメモリ読み出しで実現できる．4) の操作は本来，2) が終了し，相手の存在が確定してから行われるものであるが，副作用がないため，投機的に実行することができる．その場合，キャッシュミスが起これば，(2) 同様，相手の存在が確認されるまで待つのが効率的である．

上記で述べた操作は，並列化が可能である．その場合の同期の手順を，まず同期に成功した場合から記す．

クロック1：1) と 4)（投機実行）の同時実行
クロック2：2) と 4)（投機実行）の同時実行
クロック3：3) と 4) の同時実行
クロック4：必要ならば引き続き 4) の実行

次に，同期に失敗する場合を記す．

クロック1：1) と 4)（投機実行）の同時実行
クロック2：2) と 4)（投機実行）の同時実行
クロック3：3) と 5) の同時実行
クロック4：必要ならば引き続き 5) の実行

マルチウェイ・ジョインの場合のカウンタの更新と分岐は，2入力マッチングの場合のフラグの更新と分岐よりも時間がかかる．したがって，2入力マッチングではほとんど時間を要しなかったフラグの更新に1クロックを要することになる．

もしもメッセージデータが1語より大きかった場合（典型的には複数のデータが1つのメッセージによって運ばれてきた場合），5) には2サイクル以上を要する．この場合，データ書き込みはクロック3か，あるいはクロック4以後にも行われる．

(4) シグナル同期

数多くのトークンの間でシグナル同期をとる操作は，マルチウェイ・ジョインの操作と似ているが，データメモリにアクセスする必要のない点が異なる．すなわち，シグナル同期の場合，マルチウェイ・ジョインの1)，2)，3) を直列に処理し，同期に成功すれば（メッセージのデータを読み書きすることなく）当該スレッドを起動することになる．

(5) カウンタのオーバフロー

マルチウェイ・ジョインやシグナル同期の場合，同一地点でジョインの対象となるメッセージの数が多すぎてカウンタのオーバフローが起こる可能性がある．この場合，ジョインの木を作って対処することになる．ジョインの木においては，各ノードはカウンタをオーバフローさせない最大数のジョインをとり，データをメモリに書き込んで，次のステージのノードにメッセージを送ることになる．

ここで，フィールドとしてのカウンタの大きさ，すなわちフラグの大きさが問題となる．フラグが小さすぎると，ジョインの木が高くなり，オーバヘッドが大きくなる．フラグが大きすぎると，メモリ効率が悪くなる．たとえば，マルチウェイ・ジョイン用のフラグのフィールドが4ビットの大きさであれば，16ウェイのジョインが1つのノードでできることになり，2階層の木によって256ウェイのジョイン

ができることになる．通常の場合，4ビットの大きさで十分と考えられるが，これはより実際的な検討を加えて決定することが必要である．

ここで述べた同期では，命令をいっさい使っていない点に注意が必要である．Post，Start，Join，Nextといった命令ないし疑似命令[13]はここには存在しない．これによって，プロセッサの実行サイクル数が短縮される．また，命令を用いないことにより，あるスレッドにおける同期は，他のスレッドの実行と重畳化できるようになり，システムのスループットが向上する．また，命令フェッチおよびデコードと同時に同期が実行できるようになり，ターンアラウンドの短縮につながる．

上記の最適化された同期機構の利点をまとめると次のようになる．

(1) 効率

ここで提案した同期方式は，次のような点で効率的である．

(a) 本方式は，高度に並列化され，パイプライン化された同期機構を提供する．すなわち，1) 命令のフェッチおよびデコードと同期の並列実行，2) データメモリへのアクセスとフラグメモリへのアクセスの並列実行，3) 複数スレッドの間のパイプライン処理，がこれである．

(b) 本方式は，5種類の同期機構を提供し，一般に想定される同期に効率的に対応する．

(c) 同期のために命令を必要としないため，ターンアラウンドの短縮や同期とスレッド実行の重畳化が可能となる．

(2) 柔軟性

提案した方式は，5種類の同期を提供しており，この中で特にカウンタを用いる方式は，複雑な同期に対して柔軟に対処できる．

(3) 単純さ

メッセージが命令番地とマッチング用の番地を持つことで同期の機構が単純化されるため，デバグやトレースが他のデータ駆動型の方式と比較して容易になる．

3. パイプライン設計

前章では，マルチスレッド計算機のための高速・高機能な同期機構を提案した．本章では，前章で述べた高速な同期機構を有効に活用し，かつメッセージ処理，スレッド起動，メッセージ送信などの一連の処理を円滑に進められるような，マルチスレッド計算機のパイプライン設計について述べる．

3.1 マルチスレッド計算機のパイプライン

3.1.1 サーキュラ・パイプライン

第一世代のデータ駆動計算機やHEP[1]は，複数のスレッドが1つのパイプラインを共有し，1クロックごとに各ステージのスレッドが切り替わるという構造を持っていた．この型のパイプラインをサーキュラ・パイプラインと呼ぶ[3]．

サーキュラ・パイプラインの利点として，以下のものがあげられる．

- 1つのパイプラインで，最大限の並列アクティビティを利用できる．コンテクスト切替えのオーバヘッドがない．
- 相互結合網から命令実行系へ，さらに命令実行系から相互結合網までを1つのパイプラインでつなぐことで，計算と通信を自然で効率的な形で結合できる．
- 複雑なインタロック機構や分岐予測機構を持つことなく高いスループットを出すことができる．

これらの利点のために，サーキュラ・パイプラインは並列処理の優れた機構であると考えられてきた．2つの代表的なデータ駆動計算機であるSIGMA-1[4]とMonsoon[9]はこの型のパイプラインを持っている．

しかし，純粋なサーキュラ・パイプラインには，以下に述べるような欠点がある[8]．

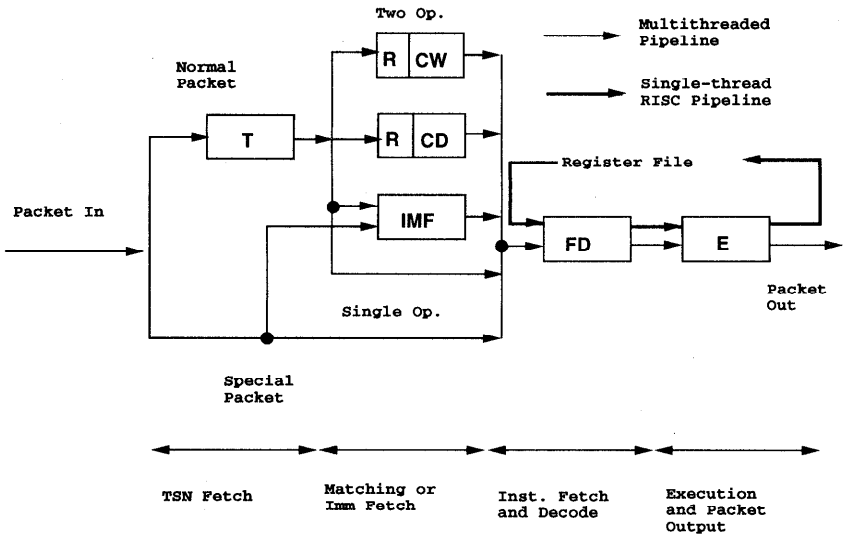
- サーキュラ・パイプラインは，先行制御を行わない．
- 並列度が低い場面では，サーキュラ・パイプラインは空きスロットが多く発生する．
- サーキュラ・パイプラインのスループットは，データの大域的な移動速度によって制約を受ける．具体的にはパケットの流れる速度と外部メモリのアクセス速度によってパイプラインの速度が決まる．
- 一般に，資源管理を行うときには不可分命令列のためにパイプラインをロックすることが必要で，サーキュラ・パイプラインは他のスレッド群との重畳化ができず，効率が悪くなる．

すなわちサーキュラ・パイプラインは，本質的に局所的で直列的な処理や，並列度が低いプログラムの実行には不向き，といえる．

3.1.2 融合型パイプライン

上述の欠点を解消することが，マルチスレッド計算機を提案する理由の1つであった[8]．典型的なマルチスレッド計算機は，RISC的なフォンノイマン型パイプラインとサーキュラ・パイプラインを組み合わせて持っている．

図2はEM-4[7]のパイプラインを図示している．EM-4のプロセッサは，2レベルに階層化されたパイ



T: TSN Fetch, R: Matching Read, CW: Check and Write
CD: Ceck and Delete, IMF: Immediate Fetch, FD: Fetch and Decode
E: Execute

図2 EM-4のパイプライン構成
Fig. 2 Pipeline organization of the EM-4.

プラインを持つ．上位階層がパケット処理を基本としたサーキュラ・パイプラインであり，これが下位階層のRISCパイプラインを包含する形となっている．前者がマルチスレッドのパイプライン処理を従来のデータ駆動計算機以上の効率で行い，後者がレジスタデータの処理を基本とする先行制御のパイプライン処理をRISCパイプラインと同程度の効率で行う．さらに，EM-4のマルチスレッドパイプラインは，パケット処理のレーテンシを小さくするために，いくつかのバイパス路を持っている．

図2では，細線がサーキュラ・パイプラインを，太線がRISCのパイプラインを示す．EM-4では，メッセージ（パケット）入力にともなってサーキュラ・パイプラインが起動される．本パイプラインでは，まず，TSN（2章参照）がフェッチされ，続いて，1クロックでマッチングが行われる．マッチングは，(1) 待合せ番地の内容を読み出し（図のR），(2) 相手の在不在をチェックし，相手が存在すればフラグを消去し（CD），相手が存在しなければパケットのデータを当該番地に書き込む（CW），という手順である．相手が即値データだった場合は，本ステージでこれが読み出される（IMF）．マッチングに成功すると，これに連続して継ぎ目なくRISC型のパイプラインが起動される．本パイプラインは，命令フェッチおよびデコード（FD），命令実行（E）の2段からなる．

*T[13),15),24)]も，RISC型のパイプラインを内蔵する．*Tプロセッサは，もともとはモトローラ社のMC88110（ピーク性能100MFLOPS）の拡張を考えており，現在ではIBM社のPower PCを想定している．*Tにおいては，メッセージ処理と同期には専用のパイプラインステージを設けてはいない．それらはもとのプロセッサチップの命令（旧版では新たに追加した特殊命令）の組合せとして実現されており，スーパスカラ的に実行される．そのためにターンアラウンド時間は長くなるが，商用機との連続性が良いのが利点といえる．

3.2 高度化されたパイプライン

3.1節の検討によって，マルチスレッド型プロセッサのパイプラインは2種類のパイプラインの組合せないし融合であるべきだと述べた．1つはマルチスレッド型のサーキュラパイプラインであり，もう1つはRISC型の単一スレッドの先行制御型パイプラインである．前者は2.3節で述べたような最適化されたデータ駆動型の同期を行うステージを組み込むことで，高速化される．

ここでマルチスレッド計算機用に最適化されたパイプラインを提案する前に，このようなパイプラインの要件を明らかにする必要がある．以下にこれを述べる．

(1) マルチスレッド実行に対しても，シングルスレッド実行に対しても高いスループットをあげること．ふつうスループットは1クロックあたりに「結果」を出す能力ではかられる．ここで考えているアーキ

テクチャでは，マルチスレッド実行のスループットは，メッセージ処理（受信・スレッド起動・送信）のレートであり，シングルスレッド実行のスループットは，クロックあたりに命令がいくつ実行されたか，という値である．

(2) 両方のパイプラインともにターンアラウンドが短いこと．

マルチスレッドのパイプラインにおいては，メッセージの入力から結果メッセージの出力までの時間が短いこと，シングルスレッドのパイプラインにおいては，命令フェッチから結果格納までの時間が短いことが重要である．シングルスレッド実行においては，ターンアラウンド時間は，通常のRISC型（スーパスカラ型）プロセッサと同等かそれ以下でなくてはならない．

最初の要求を満足させるため，パイプラインピッチは十分に短くなければならない．2番目の要求を満足させるために，コストに無理がかからない範囲でプロセッサ内の処理の並列化を進めたり，パイプラインのバイパスを行わなくてはならない．

まず，キャッシュの読み出しと書き込みはパイプライン化されるべきである．2.3節で述べた同期方式を採用した場合，プロセッサ内には3つの独立したキャッシュがあることになる．これらの中で，フラグキャッシュについては，フラグ読み出しのステージ，フラグ書き込みのステージがある．データキャッシュについては，データ読み出しステージ，データ書き込みステージ，そして場合によってはロード/ストアの命令実行ステージがある☆．パイプラインを完全に動作させるためには，フラグキャッシュは2ポート構成をとり，フラグの書き込みと次のフラグの読み出しを重畳化するのが望ましい．同様に，データキャッシュに関しても2ポート化ないし3ポート化して，同期部におけるデータ書き込みと次のデータ読み出し，さらには演算実行部による読み書きの重畳化を行うのが望ましい．

次に，演算実行部についても，現在のスーパスカラプロセッサに見られるような高度な並列化とパイプライン化がなされるべきである．スーパスカラ動作に関していえば，整数演算器・浮動小数点演算器・メモリ読み書き回路などの間だけでなく，メッセージ生成回路に独立のパイプラインを設け，これを従来のパイプラインと並列に動作させるようにするべきである．

さらに，同期機構の最適化で述べたように，同期と命令のフェッチ・デコードは並列化されるのが望ましい．

シーケンサは各処理パイプラインができるだけ非同期に処理を行えるような機構とし，必要に応じてパイプライン間の同期を最適なタイミングでとることになる．また，キャッシュなどの資源への要求が複数のパイプラインから出たときに，最適なアービトレーションを行う必要がある．たとえば，遠隔メモリ操作やカーネルによるメモリ設定などはプライオリティの高い作業であり，アービトレーションのときに優先されなければならない．

マルチスレッド計算機の要素プロセッサ用に最適化されたパイプラインを図**3**に示す．

到着したメッセージは，プロセッサがビジー状態であればいったん入力キューに格納される．そうでなければ，プロセッサは即座に，以下のようなパイプライン処理を開始する．

前半のステージ（群）では，2.3節で述べた同期と命令フェッチ・デコードが実行される．そこでは，フラグ操作・命令フェッチとデコード・データの読み書きの3つが並列化される．さらに，同期の型に応じてバイパスもある．たとえば，1入力命令を発火する場合は，データメモリの操作・フラグメモリの操作はすべてバイパスされる．

同期が成功した場合，メッセージのデータがレジスタに送られ，また，マッチングの相手のデータがメモリから読み出されてレジスタに格納される．

引き続いて，後半のステージ（群）では，スーパスカラプロセッサのパイプライン群が用意されていて，メッセージによって起動されたスレッドの命令が実行される．最初の命令（群）のフェッチ・デコードは前半のステージ（群）で行われていたが，そのデコードステージと重畳化して次の命令（群）のフェッチが始められる．さらに，最初の命令（群）の実行ステージと重畳化して3番目の命令（群）のフェッチが行われる．以下，同様にしてスレッド処理が進行する．メッセージ生成用には独自のパイプラインが用意されており，生成されたメッセージは相互結合網がビジーでないかぎり即座に網に送出される．結合網がビジーの場合は，いったん出力キューに格納される．

上記において，到着したメッセージに対する同期・命令フェッチ・デコードと現在処理中のスレッドの命令実行は重畳化されている．すなわち，メッセージが複数のデータを運んできた場合，命令の実行はすべてのデータをレジスタに格納することなく，必要なデータがそろった時点でこれを行う．このように，スーパスカラに使われるスコアボード機構は，提案するパイプ

☆ オペランドセグメントは，同期に用いられるばかりでなく，作業領域としても使われる．そのため，データキャッシュは同期部だけでなく，演算実行部によっても読み書きされるのである．

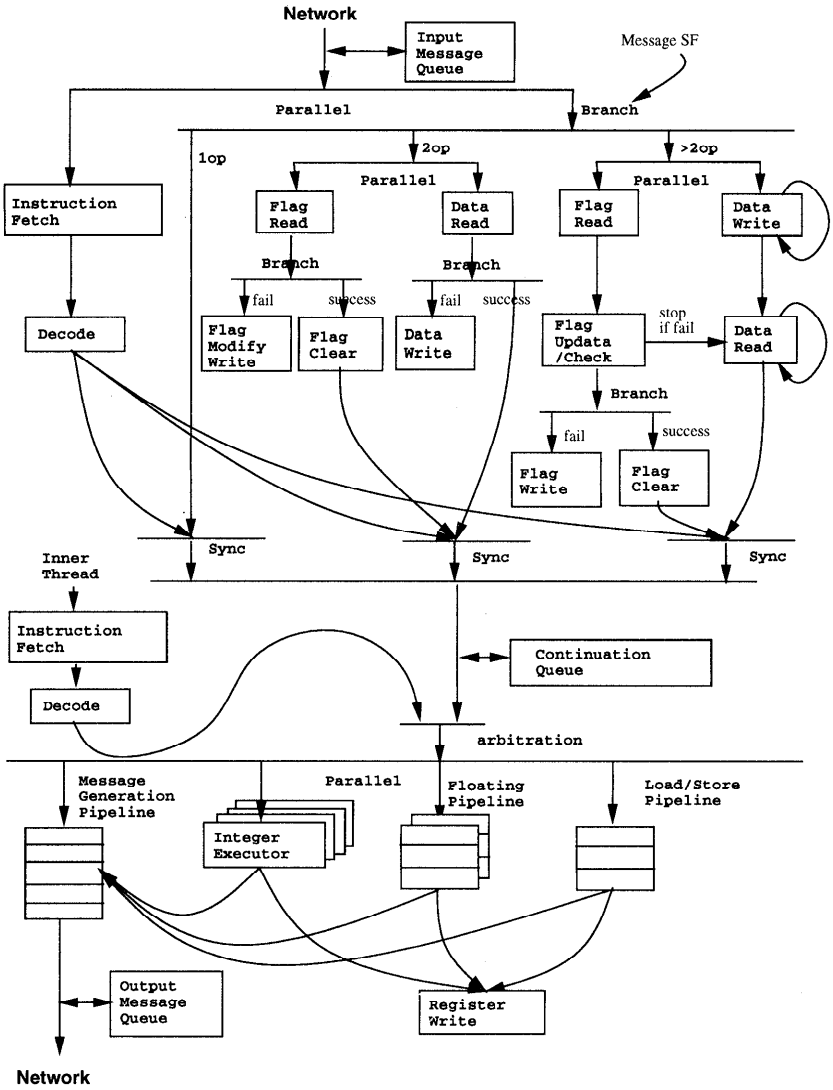


図3 最適化されたマルチスレッド型パイプライン
Fig. 3 Optimized multithreaded pipeline.

ライン機構においては，メッセージ処理およびスレッド起動にも使われる．

本方式においては，メッセージの最短ターンアラウンド時間（メッセージが入力されてから結果メッセージが出力されるまでの最短時間．スレッドの最初の命令がメッセージ出力を行う命令だった場合）は，1入力命令の場合で4クロック，2入力命令で4クロック，一般に s 入力命令の場合で $4+(s-2)$ クロックである．プロセッサのスループットは，最大で1クロック n 命令（n はスーパスカラ動作するパイプラインの本数）であり，メッセージの流れるスループットは，1クロックで1メッセージ語となる．

図3には，3つのキューが示されている．Input Message Queue，Continuation Queue，Output Message Queueがこれである．これらのキューは，それぞれ相互結合網と同期部，同期部と命令実行部，命令実行部と相互結合網の間の速度ギャップを吸収する．1つないし2つのキューを省略することも，各部のスループットのバランスに応じて考えられる．

4. 基礎的評価

4.1 基本動作速度

2.3節で提案した同期方式の基本動作速度を**表2**に示す．本方式の同期は，フラグの値に応じて2ないし

表2 提案する同期方式の基本動作速度
Table 2 Basic execution time of the proposed synchronization method.

同期の種類	同期実行時間* (clocks)	パイプライン実行時間 (clocks)
1入力（同期なし）	0	0
2入力	2	1
multiway join（成功）	3	1
multiway join（失敗）	3	1
シグナル同期	3	1

* メッセージのパイプライン段間転送時間を1クロックとした値．一般に n クロックのときは，これに $(n-1)$ を加えた値となる．

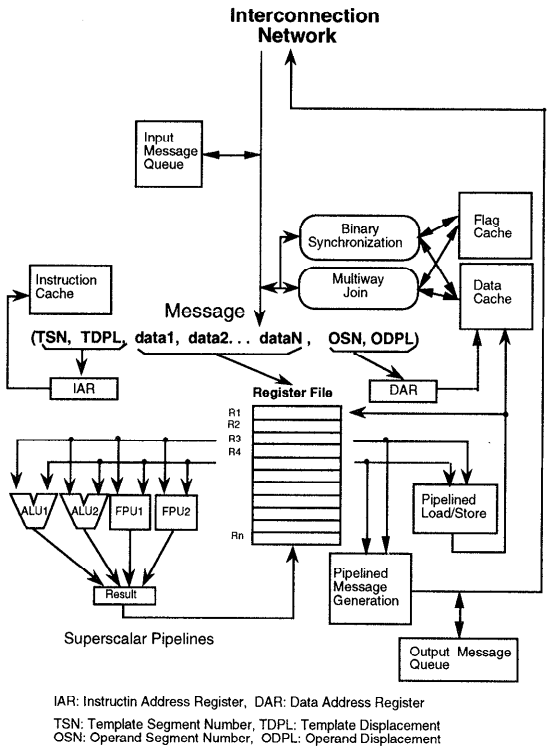


図4 本論文で提案する機構を持つプロセッサアーキテクチャ
Fig. 4 Processor architecture with the proposed mechanisms.

3クロックの時間を要するが，パイプライン動作するために，実効的な処理時間は1クロックとなる．

次に，**図4**に本論文で提案された同期機構およびパイプラインをもったプロセッサアーキテクチャの一例を示す．本アーキテクチャの基本動作は，3.2節で述べたとおりだが，ハードウェア設計の観点から見た場合，以下のような特徴を持つ．

(1) Input Message Queue, Output Message Queueという2つのハードウェアを持つ．なお，ここではハードウェアの簡単化のため，Continuation Queueは独立ハードウェアとしては設けていない．

(2) 同期用に，Binary Synchronizationの回路，Multiway Joinの回路およびFlag Cacheを持つ．シグナル同期は，Multiway Joinの回路を用いて実現される．さらに，Flag Cache，Data Cacheが多ポート化されている．

(3) メッセージ生成のための独立したパイプラインを持つ．

(4) スレッドの起動は，メッセージ中の各フィールドの値を以下のようにレジスタに挿入することで自然に行われる．

- IAR ← (TSN, TDPL)
- DAR ← (OSN, ODPL)
- register file ← data1, data2, data3, ... dataN

ただし，TSN，TDPL，OSN，ODPLなどは，2.1節で述べたものである．

4.2 実効性能に関する基礎的評価

本節では，これまでに述べた同期およびパイプライン方式の基本性能を見積もる．

スレッドの実行時間 T は，一般に式(1)で与えられる．

$$T = T_t + T_{input} + T_{sync} + T_{output} \quad (1)$$

ここで，T_t は通信・同期以外の正味のスレッドの実行時間（粒度の指標となる），T_{input} はスレッド開始前のメッセージ受信に要する時間，T_{sync} は同期に要する時間，T_{output} はメッセージの送信に要する時間である．T_{input} 以下は，動作が演算処理と重畳化される場合には，パイプライン実行時間となる．

T_{input} は，次の式で与えられる．

$$T_{input} = T_{in} \sum_{i=1}^{n} i p_i \quad (2)$$

ここで，T_{in} は，1回のメッセージ受信に要する時間であり，p_i は，i 入力の同期が起こる確率（$\sum_{i=1}^{n} p_i = 1$）である．n は，対象とする計算機の同期で実現可能な最大の入力数を表す．

次に，T_{sync} は次の式で与えられる．

$$T_{sync} = \sum_{i=2}^{n} (i-1) p_i T_{fail}(i) + \sum_{i=2}^{n} p_i T_{succeed}(i) \quad (3)$$

ここで，$T_{fail}(i)$ は，i 入力の同期が失敗した場合に要する時間，$T_{succeed}(i)$ は，i 入力の同期が成功した場合に要する時間である☆．

☆ 本スレッドを起動する前に，$i-1$ 回同期に失敗し，最後の1回で同期が成功する点に注意．

最後に，T_{output} は，次の式で与えられる．

$$T_{output} = N_{output} T_{out} \quad (4)$$

ここで，N_{output} は，スレッドあたりの平均出力メッセージ数，T_{out} は，1回のメッセージ送信に要する時間である．

以下では，時間の単位をクロックとする．本論文で提案する方式では，$T_{in} = 1$，$T_{fail}(i) = 1$，$T_{succeed}(i) = 1$，$T_{out} = 1$ である．したがって，本方式のスレッド実行時間 $T_{proposed}$ は，次の式で表される．

$$T_{proposed} = T_t + p_1 + 2\sum_{i=2}^{n} ip_i + N_{output} \quad (5)$$

比較のために，一般的な商用並列計算機のスレッド実行時間 $T_{conventional}$ を概算する．T_{in} と T_{out} は数千クロックが一般的である（たとえば CM-5 では，$T_{in} = 2800$ $(86\,\mu s)$[14]）．ここでは簡単のために，$T_{in} = T_{out} = 1000$ とする．次に同期については，不可分命令を用いてこれを実現するために，通常の命令実行よりも時間を要する．キャッシュの仕様などによって実行時間が異なるが，ここでは，$T_{fail}(i) = T_{succeed}(i) = 20$ と仮定する．以上により，$T_{conventional}$ は次の式で与えられる．

$$T_{conventional} = T_t + 1000p_1 + 1020\sum_{i=2}^{n} ip_i + 1000N_{output} \quad (6)$$

さらに，提案した方式から同期機構だけを除いた計算機（同期は命令で実行する）を考える．すなわち，$T_{in} = 1$，$T_{fail}(i) = T_{succeed}(i) = 20$，$T_{out} = 1$ とする．これは，メッセージの送信，受信およびスレッドの起動は提案したパイプラインで行うもので，同期性能が全体性能に及ぼす影響を知るためである．このときのスレッド実行時間 $T_{non-sync}$ は，

$$T_{non-sync} = T_t + p_1 + 21\sum_{i=2}^{n} ip_i + N_{output} \quad (7)$$

以上の式を用いて，以下の4つの場合について，スレッド実行時間を評価する．

［Case 1］1メッセージ入力1メッセージ出力が支配的な場合

1つのスレッドが，平均的に1つのメッセージによって起動され（同期はめったに起こらない），実行中に平均1つのメッセージを出力する場合を考える．$p_1 = 0.9$，$p_2 = 0.09$，$p_3 = 0.009$，$p_4 = 0.001$，$N_{out} = 1$ とする．

［Case 2］多入力同期の頻度が高い場合

メッセージ出力については［Case 1］と同じだが，同期が頻発する場合を考える．$p_1 = 0.25$，$p_2 = 0.25$，$p_3 = 0.25$，$p_4 = 0.25$，$N_{out} = 1$ とする．

［Case 3］メッセージ出力の頻度が高い場合

同期ついては［Case 1］と同じだが，メッセージを多く出力する場合を考える．$p_1 = 0.9$，$p_2 = 0.09$，$p_3 = 0.009$，$p_4 = 0.001$，$N_{out} = 10$ とする．

［Case 4］多入力同期とメッセージ出力がともに頻度が高い場合

$p_1 = 0.25$，$p_2 = 0.25$，$p_3 = 0.25$，$p_4 = 0.25$，$N_{out} = 10$ とする．

それぞれの場合についてのスレッド実行時間を，**図5**，**図6**，**図7**，**図8** の各グラフに記す．グラフは，横軸に T_t（粒度），縦軸に T/T_t（相対スレッド実行時間）をとり，$T_{proposed}/T_t$，$T_{conventional}/T_t$，$T_{non-sync}/T_t$ をプロットしたものである．

以下に評価結果をまとめ，考察を加える．

(1) T_t が小さいとき，すなわち細粒度のときに，提案した方式が高い優位性を示す．実行時間は，一般的な並列計算機と比較して，$T_t = 1000$ で $\frac{1}{2}$ から $\frac{1}{10}$ 程度，$T_t = 100$ で $\frac{1}{20}$ から $\frac{1}{100}$ 程度，$T_t = 10$ で $\frac{1}{150}$ から $\frac{1}{500}$ 程度で済む（図5～図8）．

(2) 多入力同期の頻度が高い場合には，そうでない場合と比較して，一般的な商用計算機の性能悪化が顕著である．たとえば，$T_t = 100$ の場合，スレッド実行時間は2倍近くになる．これは，同期そのもののオーバヘッドもあるが，複数のメッセージを受信することによるオーバヘッド（T_{input}）の増大によるところが大きい．これに対して提案した方式の性能悪化は，$T_t = 10$ で30%程度であり，T_t が100以上になると，ほとんど悪化しないことが見てとれる（図5，図6）．

(3) 同期機構を導入する効果は，粒度が小さいときに顕著である．すなわち，［Casc 1］の場合で T_t が20以下で18%以上のスレッド実行時間の短縮をもたらしている（図5）．［Case 2］［Case 4］のように，同期が頻繁に起こる場合の本機構の効果はさらに顕著であり，T_t が100のときでも，40%程度のスレッド実行時間の短縮をもたらす（図6，図8）．

(4) メッセージ出力の頻度が高い場合，一般的な計算機と比較して提案した方式は，$T_t = 10$ で $\frac{1}{500}$，$T_t = 10000$ でも $\frac{1}{2}$ のスレッド実行時間となる．これは，メッセージ生成のパイプラインを別置した効果である（図7，図8）．

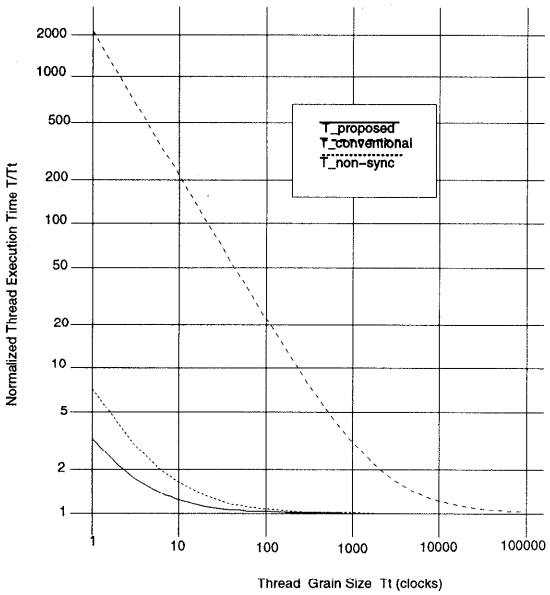


図5 スレッド実行時間の評価(1)
Fig. 5 Thread execution time (1).

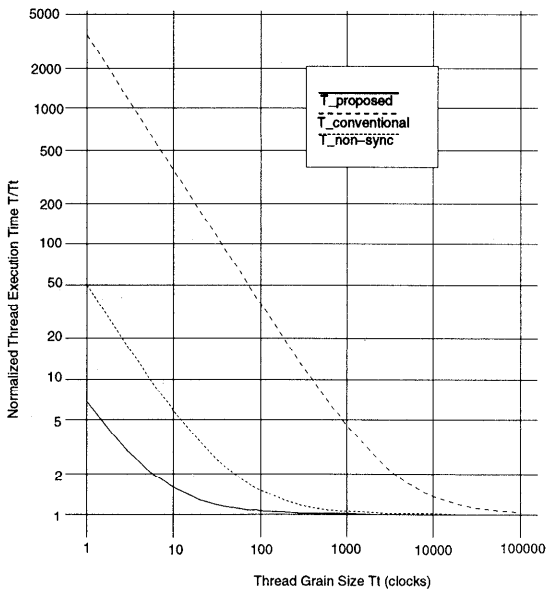


図6 スレッド実行時間の評価(2)
Fig. 6 Thread execution time (2).

以上，特に細粒度の計算や同期の頻発する計算において，提案した方式が有効であることを簡単な評価を通して示した．

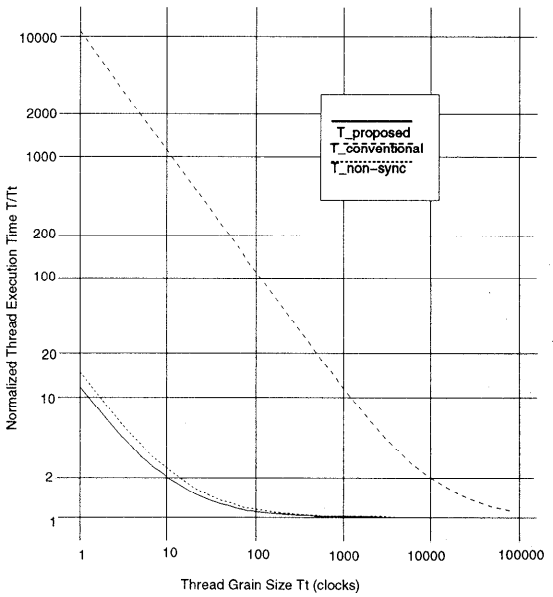


図7 スレッド実行時間の評価(3)
Fig. 7 Thread execution time (3).

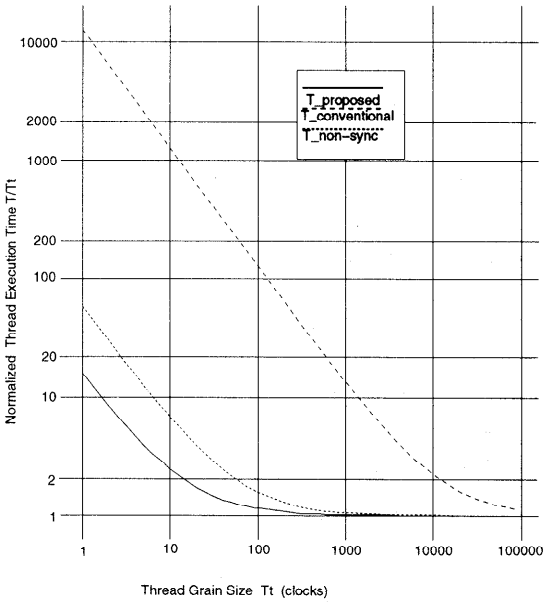


図8 スレッド実行時間の評価(4)
Fig. 8 Thread execution time (4).

5. 考　　察

5.1 アーキテクチャの最適化

5.1.1 優先度の制御

提案したパイプラインは高いスループットを提供するが，一方でスループットだけでない問題が生じる場

合もある．現在実行中のスレッドよりも別のスレッドを優先実行しなくてはならない場合などがこれである．たとえば，ある遠隔メモリ読み出しが全体の時間にとってクリティカルな場面では，もしこの遠隔メモリ読み出しのメッセージが他の長いスレッドの実行によってブロックされれば，全体のパフォーマンスは低下する．

この問題を解決するには，2つの方法が考えられる．1つは，遠隔メモリ操作を別系統の独立したハードウェアで実現する方法である．もう1つは，プリエンプションを起こす方法である．前者においては，プロセッサは優先度の高い処理のための新しいパイプラインを持つ必要があり，そのために新たなレジスタやデータパスが必要となる．*T では当初 I-Structure の操作にこの方式を使うことにしていた[13]．後者においては，プロセッサは高速のコンテクストスイッチ機構を持つ必要がある．3章で述べた最適化されたパイプラインを有効に利用するためには，コンテクストスイッチのオーバヘッドは0クロックか1クロック程度に抑えなければならない．文献18)で述べられた命令挿入機構は，複数のレジスタセットを設けて高速のコンテクストスイッチを行うものであり，利用できると考えられる．

どちらの方法を用いる場合でも，プロセッサ内部には，効率的なプライオリティキューが必要であり，場合により相互結合網のスイッチの内部にもこれが必要となる．

5.1.2 より柔軟性の高いスレッド・インタリーブ方式

これまでの議論では，命令実行部の各パイプラインは，同時には，1つのスレッド内の命令によって利用されることを想定していた．しかし，より細かなスレッド制御を行うことで，パイプラインのバブルを減らし，性能向上をはかることが考えられる．M-Machine[17]におけるプロセッサ・カップリングなどがこれにあたる．

一例として，複数のスレッドが浮動小数点のパイプラインを同時に共有して用いる場合が考えられる．最も極端な場合では，複数のスレッドの任意の命令がパイプライン群の任意のスロットを使えるような構成も考えられる．このような処理方式を，スレッド・インタリーブと呼ぶ．

スレッド・インタリーブは，ハードウェア資源（レジスタなど）の増加の問題やシーケンサの複雑さの問題，割込みや例外処理の複雑さの問題などから，高度化することが必ずしも有利とはいえない技術と考えられる．

5.2 コストパフォーマンス

4.2節の評価から分かるように，1つ1つのスレッドが大きなプログラムでは，単一スレッドの実行中，命令は，ほとんどの時刻でパイプライン実行され，同期に要する時間の全体に占める割合は小さい．このような場合，本論文で述べられた同期方式やパイプライン構成は過剰デザインとなる可能性がある．

しかし，ハードウェア量の点からは，これらの機構にはほとんど追加コストがかからない．EM-4 の直接マッチングは，わずかに 3,610 個の CMOS のゲートが必要であったにすぎない[7]．これは，整数演算ユニットのような他の回路と比較して，ほぼ無視できる大きさである．提案した機構は，EM-4 と比較するとわずかに多くのハードウェアを必要とする．というのも，パイプラインピッチは EM-4 の約半分の長さであり，そのためにパイプラインレジスタの追加が必要になるからである．しかし，これは VLSI の中に占める割合からいえば，問題ではない．

本論文で述べた機構のための論理回路は，単純なものである．複数のスレッドの間の順序関係の調整を行うインタロックが基本的に存在しないからである．さらに，エラー処理や例外処理を除き，通信・同期などにいっさいの割込み処理の機構を必要としないことも，単純さに寄与している．

マルチポートキャッシュ，複数本のバス，複数個のデコーダ，アービタの回路などは，全体のハードウェア量を大きくする原因となっている．しかし，これらはよく知られた単純な回路である．また，レジスタからのバスはすべてチップ内に納められるので，VLSI のピン制限の問題は生じない．

最後に，相互結合網とメモリのコストの問題を考える必要がある．提案した同期方式では，同期のために余分のフィールドが必要となる．ETS と比較した場合，SF と ODPL の2つのフィールド分が大きくなっている．しかし，マルチスレッド計算機においては，Monsoon などのデータ駆動計算機と比較して，同期の頻度は小さいため，同期に要するアドレススペースが小さくてよい．そのため SF と ODPL のフィールドは小さくてよいと考えられる．具体的には，SF は2.3節の検討により4ビット程度でよいと考えられる．

命令メモリをアクセスするための TDPL も同様の理由で小さくなる．OPDL，TDPL の大きさについては，一般の並列プログラム群の性質を見て設定することを検討している．

さらに，プロセッサ内で並列性を利用するために，本アーキテクチャでは，演算実行部に要する時間が短

縮される．そのため，パフォーマンスを出すためには，パイプラインのターンアラウンドを短くすることが必要となり，したがって，同期部のターンアラウンドを短くすることが必要となる．

以上の考察から，提案した同期方式とパイプライン構成は，マルチスレッド計算機のプロセッサとして，VLSIによる実装性が高く，同時にコストも妥当であると考えられる．

5.3 関連事項

ここでは，局所的なデータ駆動型の同期について検討したが，マルチスレッド計算機の同期を考える場合には，大きな構造データをI-structureやM-structureとして処理する機構についても最適化が必要である．さらに，バリアなどの大域的同期についての最適化も必要である．これらのうち，前者についてはフラグ操作に続くデータのリンクが複雑であり，本論文で述べた方式によるフラグ操作の後に，割込みによってソフトウェア的処理をすることが有利である[19]．後者については，本論文で述べた同期をカスケード状に組み合わせたもので対処する方式が考案されており，EM-4の実験では十分な性能が得られている[20]．

別の重要な問題として，キャッシュの構造の最適化がある．キャッシュミスが頻発する場合では，パイプラインは効率良く動作しない．スケジューリングの技術によって並列度を下げることなくヒット率を上げることがどれぐらい可能か，検討されなければならない．

さらに，コヒーレントキャッシュを本方式と組み合わせ，レーテンシの短縮をはかること[24]も検討対象である．

効率の問題以外でも，現実の計算機では，仮想化を含むネーミングの機構が実装されなければならない．著者らはその方式を検討中[21]であり，別途報告する．仮想化を行った場合，物理的なデータ配置がシステム性能に大きな影響を及ぼすため，データ配置の方法も，個々のプログラムごとに検討されなくてはならない．

ここで述べた同期方式とパイプライン構成を基礎として，著者らは超並列計算機RWC-1を開発中である[22]．

6. おわりに

本論文は，マルチスレッド型超並列計算機の2つの基本機構である同期とパイプラインについて提案を行った．ここで提案した同期機構は，直接マッチングを基本とし，高度に並列化・パイプライン化されたものである．1RISCクロックごとに1度の同期が可能であり，同期に要する時間は，相手がキャッシュの中にあれば，通常2ないし3クロック程度である．

ここでは，提案した同期機構を用いて，パイプラインを高速化することを考え，マルチスレッド型のプロセッサ内の最適化パイプラインを提案した．本パイプラインにおいては，さまざまな動作が高度に並列化・重畳化される．たとえば，同期と命令フェッチ・デコードが並列実行され，通常の命令実行とメッセージ生成が並列化される．さらに，パイプラインピッチはキャッシュサイクル程度に短縮されている．したがって，1プロセッサのターンアラウンド時間も短縮され，メッセージが到着してから結果のメッセージが出力されるまで，最短で4クロックとなっている．

本論文では，さらに，提案した方式の基礎的評価を行い，特に細粒度の計算において，本方式が従来方式と比較して，1桁から2桁程度の性能改善をもたらす可能性があることを示した．

これからの課題を以下に列挙する．

(1) ソフトウェアシミュレータと実用規模のベンチマークを用いて，提案した方式のより細かな定量的評価をすること．

(2) 優先度の扱い，コヒーレント機構との組合せ，負荷分散，粒度最適化，データ分配，I/Oなど，関連する問題を含めた検討を進めること．

(3) プロトタイプマシンの構築によるコストパフォーマンスの評価と性能評価．
プロトタイプマシンとして，RWC-1[22]を開発している．本マシンに関しては，別途報告する．

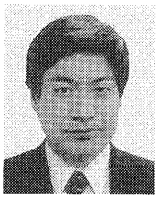
謝辞 本研究を遂行するにあたりご指導いただいた，マサチューセッツ工科大学のArvind教授，電子技術総合研究所の太田公廣元情報アーキテクチャ部長，山口喜教計算機方式研究室長，計算機方式研究室の同僚諸氏，新情報処理開発機構の島田潤一研究所長に感謝いたします．

参考文献

1) Smith, B.J.: A Pipelined, Shared Resource MIMD Computer, *Proc. 1978 ICPP*, pp.6–8 (1978).
2) Arvind, Kathail, V. and Pingali, K.: A Dataflow Architecture with Tagged Tokens, Tech. Rep. TR-174, Lab. Computer Science, MIT (1980).
3) Gurd, J., Kirkham, C.C. and Watson, I.: The Manchester Prototype Dataflow Computer, *Comm. ACM*, Vol.21, No.1, pp.34–53 (1985).
4) Hiraki, K., Nishida, K., Sekiguchi, S., Shimada, T. and Yuba, T.: The SIGMA-1

Dataflow Supercomputer: A Challenge for New Generation Supercomputing Systems, *Journal of Information Processing*, Vol.10, No.4, pp.219–226 (1987).
5) Iannucci, R.A.: Toward a Dataflow/von Neumann Hybrid Architecture, *Proc. 15th ISCA*, pp.131–140 (1988).
6) Alverson, G., Alverson, R., Callahan, D., Koblenz, B., Porterfield, A. and Smith, B.J.: Exploiting Heterogeneous Parallelism on a Multithreaded Multiprocessor, *Proc. ICS '92*, pp.188–197 (1992).
7) Sakai, S., Yamaguchi, Y., Hiraki, K., Kodama, Y. and Yuba, T.: An Architecture of a Single Chip Dataflow Processor, *Proc. 16th ISCA*, pp.46–53 (1989).
8) 坂井修一，平木 敬，山口喜教，児玉祐悦，弓場敏嗣：データ駆動計算機のアーキテクチャ最適化に関する考察，情報処理学会論文誌，Vol.30, No.12, pp.1562–1572 (1989).
9) Papadopoulos, G.M. and Culler, D.E.: Monsoon: An Explicit Token Store Architecture, *Proc. 17th ISCA*, pp.82–91 (1990).
10) Grafe, V.G., Hoch, J.E., Davidson, G.S., Holmes, V.P., Davenport, D.M. and Steele, K.M.: The Epsilon Project, *Advanced Topics in Data-Flow Computing*, Chapter 6, pp.175–205, Prentice Hall (1991).
11) Papadopoulos, G.M. and Traub, K.R.: Multithreading: A Revisionist View of Dataflow Architecture, *Proc.18th ISCA*, pp.342–351 (1991).
12) Sato, M., Kodama, Y., Sakai, S., Yamaguchi, Y. and Koumura, Y.: Thread-based Programming for the EM-4 Hybrid Dataflow Machine, *Proc. 19th ISCA*, pp.146–155 (1992).
13) Nikhil, R.S., Papadopoulos, G.M. and Arvind: *T: A Multithreaded Massively Parallel Architecture, *Proc. 19th ISCA*, pp.156–167 (1992).
14) Eicken, T, Culler, D.E., Goldstein, S.C. and Schauser, K.E.: Active Messages: A Mechanism for Integrated Communication and Computation, *Proc. 19th ISCA*, pp.256–266 (1992).
15) Papadopoulos, G.M., Boughton, G.A., Greiner, R. and Beckerle, M.J., *T: Integrated Building Blocks for Parallel Computing, *Proc. SC93*, pp.624–635 (1993).
16) Dally, W., Chien, A., Fiske, S., Horwat, W., Keen, J., Larivee, M., Lethin, R., Nuth, P. and Wills, S.: The J-Machine: A Fine-Grain Concurrent Computer, *Proc. IFIP 89*, pp.1147–1153 (1989).
17) Keckler, S.W. and Dally, W.J.: Processor Coupling: Integrating Compile Time and Runtime Parallelism, *Proc. 19th ISCA*, pp.202–213 (1992).
18) Toda, K., Nishida, K., Uchibori, Y., Sakai, S. and Shimada, T.: Parallel Multi-context Architecture with High-speed Synchronization Mechanism, *Proc. 5th IPPS*, pp.336–343 (1991).
19) 岡本一晃，松岡浩司，廣野英雄，横田隆史，堀敦史，児玉祐悦，佐藤三久，坂井修一：超並列計算機 RWC-1 における同期機構，情報処理学会計算機アーキテクチャ研究会，101-2 (1993).
20) Shaw, A., Kodama, Y., Sato, M., Sakai, S. and Yamaguchi,Y.: データ駆動計算機 EM-4 におけるデータ並列プログラミング，並列処理シンポジウム JSPP '92，pp.179–186 (1992).
21) 松岡浩司，岡本一晃，廣野英雄，横田隆史，堀敦史，児玉祐悦，佐藤三久，坂井修一：超並列計算機 RWC-1 における記憶構成，情報処理学会計算機アーキテクチャ研究会，101-3 (1993).
22) Sakai, S., Matsuoka, H., Okamoto, K., Yokota, T., Hirono, H., Kodama, Y. and Sato, M.: RWC-1 Massively Parallel Architecture, *Proc. HPCC '94*, pp.33–38 (1994).
23) Kodama, Y., Sakane, H., Sato, M., Yamana, H., Sakai, S. and Yamaguchi, Y.: The EM-X Parallel Computer: Architecture and Basic Performance, *Proc. 22nd International Symposium on Computer Architecture*, pp.14–23 (1995).
24) Chou, D., Ang, B.S., Arvind, Beckerle, M.J., Boughton, A., Greiner, R., Hicks, J.E. and Hoe, J.C.: StarT-NG: Delivering Seamless Parallel Computing, *Proc. Euro-Par '95*, pp.101–116 (1995).

(平成 8 年 4 月 11 日受付)
(平成 9 年 6 月 3 日採録)

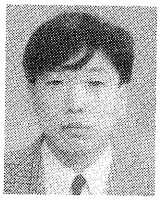


坂井 修一（正会員）

1958年生．1981年東京大学理学部情報科学科卒業．1986年同大学大学院情報工学専門課程修了．工学博士．同年，電子技術総合研究所入所．1990年同研究所主任研究官．1991年4月より1年間米国MIT招聘研究員．1993年3月より1996年2月までRWC超並列アーキテクチャ研究室室長．1996年10月より筑波大学助教授（電子・情報工学系）．システム一般，特にアーキテクチャ，並列処理，スケジューリング問題などの研究に従事．情報処理学会論文賞（1990年度），日本IBM科学賞（1991年），市村学術賞（1995年），ICCD Outstanding Paper Award（1995年）など受賞．IEEE，電子情報通信学会会員．

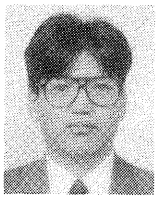
岡本 一晃（正会員）

1962年生．1986年慶應義塾大学理工学部電気工学科卒業．同年三洋電機（株）に入社．主にデータ駆動計算機を中心とする並列処理アーキテクチャの研究に従事．1992年10月より（技組）新情報処理開発機構に出向，超並列アーキテクチャの研究に従事．主任研究員．ICCD Outstanding Paper Award（1995年）受賞．



松岡 浩司（正会員）

1961年生．1984年東京工業大学工学部電気電子工学科卒業．1986年同大学大学院理工学研究科電子物理工学専攻課程修了．同年日本電気（株）に入社．1992年10月（技組）新情報処理開発機構に出向，主任研究員．現在，計算機システム全般，特にプロセッサアーキテクチャの研究に従事．ICCD Outstanding Paper Award（1995年）受賞．



廣野 英雄

1968年生．1991年筑波大学第三学群基礎工学類卒業．同年三洋電機（株）に入社．1992年より（技組）新情報処理開発機構に出向中．計算機アーキテクチャの研究に従事．ICCD Outstanding Paper Award（1995年）受賞．電子情報通信学会会員．

横田 隆史（正会員）

1960年生．1983年慶應義塾大学工学部電気工学科卒業．1985年同大学大学院電気工学専攻修士課程修了．同年三菱電機（株）に入社，知識処理向けアーキテクチャおよび並列アーキテクチャの研究に従事．1993年12月より1997年3月まで（技組）新情報処理開発機構に出向し，超並列アーキテクチャの研究に従事．ICCD Outstanding Paper Award（1995年）受賞．電子情報通信学会会員．