7N-6

高速並列処理ワークステーション（TOP－1）
－並列処理同期化機構－

小原盛幹，清水茂則

日本アイ・ビー・エム株式会社
東京基礎研究所

はじめに

マルチプロセッサにおいて効率のよい並列処理を行うためには，プロセッサ間の同期をとる機構がたいへん重要である。プロセッサが最も高速にアクセスできる媒体は，通常，メモリであり，このメモリを使ってプロセッサ間の同期が行えると都合がよい。 TOP-1 では，共有メモリ上の共有変数を使って，プロセッサ間の同期をとることを基本にしている。ただし，共有変数だけでは，非同期に発生するイベントをプロセッサ間で効率よく伝達するとはむずかしい。このため，メッセージ・パッシングを行うハードウェアも設けた。本稿では，これらの同期化機構について述べる。

共有変数による方法

共有変数を使って同期を行うためには，共有変数へのアクセスの排他制御が必要になる。 TOP-1 では，バスのロックによりこの排他制御を実現している。

Lock Prefix

80386 では，ビット演算などいくつかの演算命令の命令コードの前に Lock Prefix を付加すると，命令の実行中はバスのロックを要求する信号 (LOCK#) をアサートする。この Lock Prefix により，リード・モディファイ・ライトを使った Test-and-Set などの同期プリミティブを実現することができる。

リード・モディファイ・ライトでは，変数へのアクセスが排他的でなければならない。ところが通常のリード・オペレーションでは，キャッシュでヒットするとバスのアクセスが発生しないので，複数のプロセッサが同時にリードする可能性がある。そこで TOP-1 では， LOCK# 信号がアサートされると，キャッシュのヒット・ミスを調べる前にバスの獲得・ロックを行っている。この後，通常のキャッシュ制御シーケンスが続き，ライト・オペレーションを行った後でバスを解放する。

デュアル・バス・ロック

TOP-1 はアドレスでインターリーブされた2 つのバスを持っているので，排他制御を行う共有変数が両方のバスの領域にまたがってとられる可能性もある。この場合，注意してバスのロックを行わないと，デッドロックになる可能性がある。 TOP-1 では，次のようなシーケンスでバスのロックを行うようにした。

1. 偶数バスのアクセス権を獲得する
2. バスをロックする
3. 奇数バスのアクセス権を獲得する
4. 共有変数にアクセスする
5. バスを解放する

この方式では，偶数バス獲得の直後にバスのロックを行っているので，次のサイクルで奇数バスの獲得が保証される。このため，効率のよいバス・ロックが実現されている。

Multi-Instruction Lock

80386 の提供する LOCK# 信号を使えば，セマフォを実現することができ，プロセッサ間の排他制御を行える。しかし， enqueue/dequeue など，短いクリティカル・セクションが頻繁に現れる場合には，セマフォのオーバーヘッドが大きくなる。そこで TOP-1 では，複数の命令を実行する間，共有バスをロックする機構を実現した。

TOP-1 では，キャッシュ・バス・コントローラには I/O マップされたいくつかのレジスタがある。図 1に示したようにプロセッサがあるレジスタに出力命令を実行すると，バス・ロックやロック解除が行なわれる。バス・ロックは， Lock Prefix の場合と同様のシーケンスで，両方のバスがロックされる。

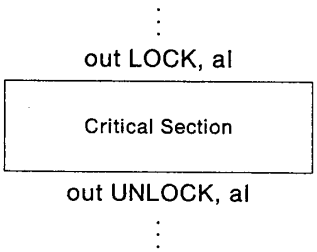

図 1: Multi-Instruction Lock

メッセージ・パッシングによる方法

プロセッサ間でイベント・ドリブンな通信を行うために，メッセージ・パッシング機構を TOP-1 に持たせた。各キャッシュ・バス・コントローラは，図 2に示したメッセージ・パッシングの制御機構を持っている。1 つのメッセージのサイズは 32 ビットで，1 つの受信バッファは1 つのメッセージをストアできる。 Destination Register は，メッセージの宛先の指定に使われる。メッセージ・パッシングには，共有メモリをアクセスするときと同じ共有バスを使用する。偶数バスと奇数バスがそれぞれ同等なメッセージ・パッシングの機構を持っている。

プロセッサは，以下のように， I/O 命令によりメッセージ・パッシングを行う。

High-Performance Multiprocessor Workstation (TOP-1)
- Multiprocessor Synchronization -
Moriyoshi Ohara, Shigenori Shimizu
IBM Research, Tokyo Research Laboratory

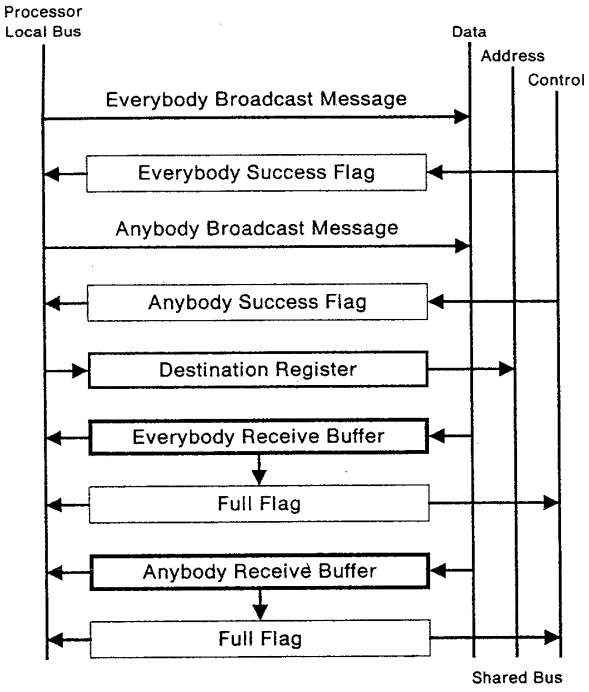

図 2: Message Passing Hardware

1. プロセッサがある I/O アドレスに出力命令を開始する。
2. キャッシュ・バス・コントローラがバスを獲得する。
3. 宛先の受信バッファが空のとき，プロセッサの出力データがバッファに書かれる。
4. バスが解放され，プロセッサの I/O 命令が終了する。

このとき，受信バッファにすでにメッセージがストアされ，プロセッサがまだ読んでいない場合には，メッセージは受信バッファには書かれず，メッセージ・パッシングは失敗する。送信側のプロセッサは，キャッシ・バス・コントローラのフラグ（ Success Flag ）を読んで，メッセージ・パッシングの失敗・成功を知ることができる

メッセージの宛先指定方式

1 対1 のメッセージ・パッシングが可能であれば，基本的な機能を提供できる。実際には，複数のプロセッサに同時にメッセージを送りたい場合も多い。 TOP-1 では，メッセージの宛先指定により汎用性のある2 種類の方式を用意した。

1 つは， Everybody Broadcast と呼ばれる。この方式では，宛先に複数のプロセッサを指定でき，指定したすべてのプロセッサがメッセージを受信できたとき，メッセージ・パッシングが成功する。1 つでもプロセッサの受信バッファが空でなく，受信できないと，メッセージ・パッシングは失敗する。このとき，メッセージはいずれのプロセッサの受信バッファにも書かれない。

また，指定したプロセッサがすべてメッセージを処理する必要がない場合もある。たとえば，ある I/O からのデータがレディーになり，I/O プロセッサ（ 5570 ）が 1 0 台のプロセッサ のいずれかにその処理を要求したいとする。この場合，すべてのプロセッサが I/O の処理要求メッセージを受信する必要はない。 Everybody Broadcast を使うと，すべてのプロセッサが受信バッファを空になるまで待たなければならない。そこで，Anybody Broadcast というメッセージの宛先指定方式を設けた。この方式では，指定した複数の宛先のうち，1 つ以上のプロセッサが受信できれば，メッセージ・パッシングが成功する。受信バッファが空のプロセッサのみが，このメッセージを受けとる。

TLB コンシステンシー

80386 はページングにおけるアドレス変換を高速に行うため， TLB (Translation Lookaside Buffer) を内蔵している。しかしながら TLB は，アドレス変換テーブルの一部のコピーを持つ，一種のキャッシュである。このため，マルチプロセッサ・システムでは，キャッシュと同様に， TLB に対してもコンシステンシーを保つ必要がある。

TLB コンシステンシーは，特にページ・スワップのとき問題になる。 TOP-1 では，図 3に示したように，メッセージ・パッシングとソフトウェアで TLB のコンシステンシーを保っている。ページ・スワップを行うプロセッサ（ Swapping Processor ）は，スワップするページの Presence Bit をクリアし，他のプロセッサが新にこのページを使いはじめないようにする。次に， Everybody Broadcast を使って，すべてのプロセッサにあるメッセージを送る。このメッセージを受信するとインタラプト（ Message INT ）が発生する。インタラプト・ルーティンは自分の TLB をフラッシュした後，共有変数を使ってフラッシュの完了を示す。ページ・スワップを行なうプロセッサは，他のすべてのプロセッサのフラッシュを確認した後，実際にスワップを始める。

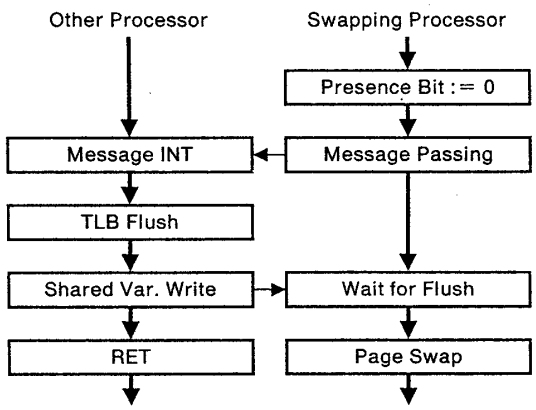

図 3: TLB Flush Algorithm

おわりに

TOP-1 で実現したプロセッサ間の同期化機構について述べた。今後は，システム・ソフトウェアの実現などを通して，これらの同期化機構を評価する予定である。