分散並列OS「Orion」の試作 高速通信機能の検討

2P−6

千葉寛之，岩嵜正明，吉澤聡，宇都宮直樹，薗田浩二，山内雅彦
(株)日立製作所　中央研究所

1．はじめに

我々は分散環境上で並列計算を可能とする分散並列OS「Orion」の研究を行っている[1]。本稿では、Orionが提供する並列計算指向の高速通信機能について述べる。

流体解析等の数値計算分野では、問題を部分領域に分割し、各ノードに部分領域を処理するプロセスを割り当てて並列に処理する手法がよく用いられる[2]。この手法では、境界部分のデータを隣接ノード（厳密には隣接する部分領域を処理するノード）へ転送する処理が必要となる。

このノード間の通信に、例えばUnixのsocketインタフェースを用いると、プロトコル処理のオーバヘッドが大きく、並列処理に必要な通信性能を実現できない。socketに限らず、多重アドレス空間やマルチタスキングを前提に信頼性のあるプロセス間通信を実現しようとすると、カーネル内でのバッファリングが不可欠となる。このバッファリングは、ユーザ空間とカーネル空間でのコピー処理を必要とし、オーバヘッドを転送データ量に比例して増大させてしまう。

Orionシステムでは、カーネル内バッファリングを排除してユーザプロセス間で直接データを転送するCombuf通信方式、及びこれを用いたプログラミング手法を検討している。

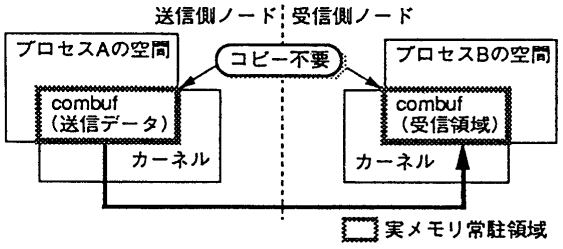

図1．Combuf通信方式

2．Combuf通信方式

Combuf通信方式は、図1に示すように送受信双方のプロセス空間上に通信用の領域（Combuf: Communication buffer）を設け、自（プロセス空間内）Combuf上のデータを他Combuf内の任意の位置に直接転送するインタフェースを提供する。Combufは各ローカルノードの物理メモリの一部をページ固定領域として設定し、ユーザアドレス空間の一部にマッピングすることにより実現する。こうすることによりデータ受信時にプロセス空間の切り替え無しに直接ユーザ空間にデータを書き込むことが可能となる。

Prototyping of Distributed-Parallel Operating System "Orion"
- Study on Inter-Nodal Direct Data Transfer Method -
Hiroyuki CHIBA, Masaaki IWASAKI, Satoshi YOSHIZAWA,
Naoki UTSUNOMIYA, Kouji SONODA, Masahiko YAMAUCHI
Central Research Laboratory, Hitachi, Ltd.

2.1．API (Application Programing Interface)

Combuf通信は以下のAPIを用いて行なう。

（1）Combuf割り当て：　通信を始める前に、各プロセスがそれぞれ同じ大きさのCombufを割り当てる。Combufは、それを用いる送受双方のプロセスが割り当てを完了したときに使用可能となる。

（2）送信：　送信は、転送データの先頭アドレスと、データサイズを指定し、相手プロセスのCombuf内の領域の先頭アドレスより始まる領域に直接データを上書き（オーバーライト）する。

尚、Combuf通信は、(1)同一の送受信プロセスの組におけるデータの送信順序と到着順序は一致する、(2)送信データは必ず到達する、の2つを前提とする。

2.2．プログラミング上の課題

Combuf通信方式は、送信時に受信側のアドレスを指定するインタフェースを採用し、カーネル内バッファリングを行わない通信を実現する。しかしその半面、バッファリングを排除したことにより、従来カーネルレベルで行なっていた処理をアプリケーションレベルで実現しなければならない。Combuf通信方式を活かすためには、以下の課題を解決しなければならない。

（1）送受信プロセス間同期：　Combuf通信方式では、受信側Combuf上に受信データが直接上書きされるため、送信側が受信領域の空きを確認してデータ転送を行なわなくてはならない。この同期制御はCombuf通信の転送性能に見合う低オーバーヘッドで実現しなければならない。

（2）演算と受信の相互排除：　上記の送受信プロセス間同期が確実に行えないと、受信側のプロセスが演算に使用しているデータ上に、受信データが上書きされることを抑止できない。そのため、同一Combuf上のデータに対する演算と転送を排他的に実行する必要がある。

3．クロスカスケード・データストリーム

以上述べたCombuf通信方式の課題を解消するプログラミング手法であるクロスカスケード・データストリーム（CCDS: Cross Cascade Data Stream）について論じる。CCDSは、分散環境では高速化が困難な全ノード一斉同期を必要としないプログラミングを実現する。

（1）アルゴリズム

CCDSは、ダブルバッファリングとデータフロー動作の組み合わせにより送受信プロセス間の同期を実現する。CCDSは、図2に示すアルゴリズムに従って各ノードの演算処理を行なう。図2の中央部分が1プロセスによる処理を表わし、斜めの矢印は左右のプロセスとの通信を表している。

CCDSでは、演算処理を２つのフェーズに分け、隣接ノードからのデータがすべて揃うまで、次の演算フェーズを開始しない。隣接ノードとデータを交換する配列の境界部分は２面（A面-B面）のダブルバッファリングを行なう。即ち、演算はA面とB面を交互に用いて行ない、A面を用いて演算を行なっている間は、隣接ノードからのデータをB面側に受信する。

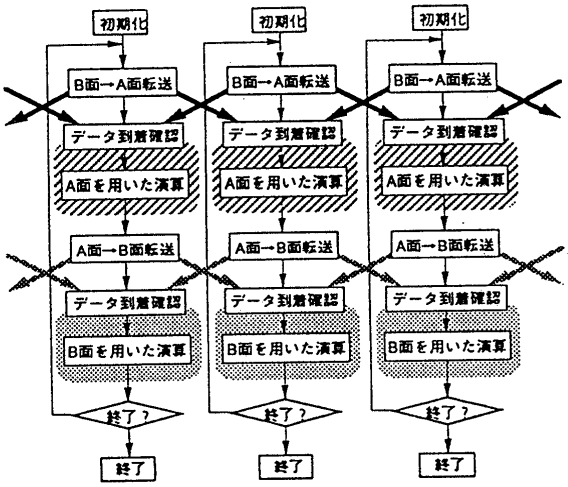

図２．CCDSのアルゴリズム

（２）送受信プロセス間同期の実現

図２の制御に従えば、あるノードの演算処理時間が伸びる等の原因により、ノード間の処理の進捗に不均衡が生じても、自動的に隣接ノードとの同期が取れる。隣接ノードは処理が遅れたノードからのデータが到着しない限り、反対側の面の演算を開始しない。この様子を図３及び図４に示す。

（３）演算と受信の相互排除の実現

図３は全ノードの処理が均衡して進行している場合を示す。各演算部分に表示する数字は、演算のフェーズを示している。図３に示すようにB面からA面へのデータ転送は、フェーズ１と２、フェーズ３と４の間で行われる。これらの転送は、フェーズ２から３へのA面からB面への逆方向のデータ転送により、上書きが生じないことが保証される。

このように、CCDSではダブルバッファリングにより、受信領域へのデータの上書きを防ぐとともに同期制御を実現できる。

（４）CCDSによる連鎖同期

図４はプロセス２のA面側の演算時間が伸びた場合を示す。プロセス２のA面側の演算時間が伸びても、隣接するプロセス１やプロセス３から送られてくるデータはB面側に書き込まれるため、演算途中のデータを破壊することはない。また、プロセス１やプロセス３は、プロセス２からのデータが到着するまで、B面側の演算を開始しない。

この様にCCDSでは一種のフィードバック作用が働くため、あるノードだけが他のノードを追い越して、次の演算フェーズに進むことがない。CCDSでは、このフィードバック作用により、全ノードに渡る一斉同期を用いなくても、隣接ノード間の同期の連鎖作用によってシステム全体に渡る同期を実現できる。

４．おわりに

カーネル内バッファリングを排除したCombuf通信方式とこれを応用したプログラミング手法であるクロスカスケード・データストリーム(CCDS)について述べた。今後、CCDSの上位インタフェースを確立していくとともに、様々な問題へのCombuf通信の適用を検討していく。

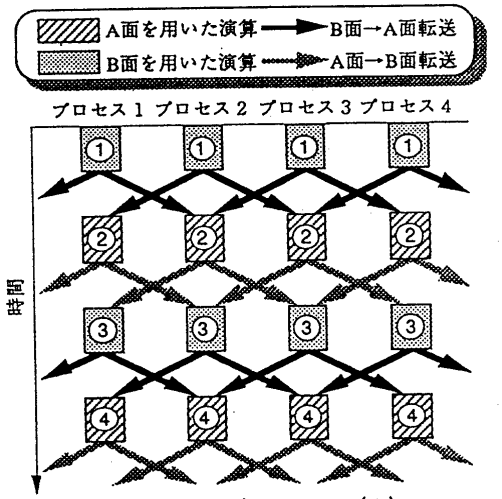

図３．CCDSの処理フロー（１）

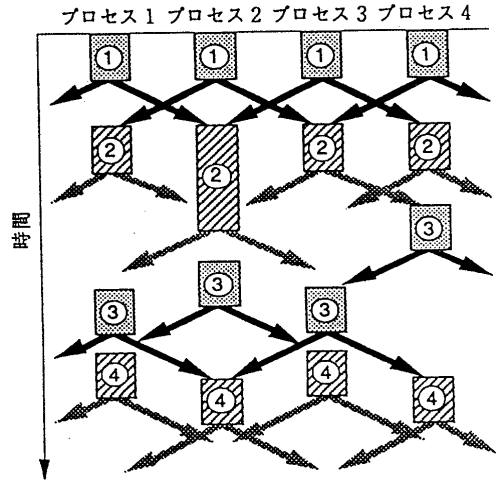

図４．CCDSの処理フロー（２）

参考文献

[1] 岩嵜, 他, 「分散並列OS「Orion」の試作ーシステムの概要」, 情報処理学会第45回全国大会予稿集, 2P-01, 1992.

[2] 吉澤, 他, 「分散並列OS「Orion」の試作ーシステムの性能評価」, 情報処理学会第45回全国大会予稿集, 2P-05, 1992.