メモリベース通信による非対称分散共有メモリ

松本 尚 † 駒嵐 丈人 ‡ 渦原 茂 ‡ 平木 敬 †

† 東京大学 大学院理学系研究科 情報科学専攻
‡ 有限会社 アックス

本稿では特殊な通信同期ハードウェアを仮定しないワークステーションクラスタや分散メモリ型並列計算機において、高速かつ仮想化され保護されたユーザレベル通信同期であるソフトウェアによるメモリベース通信（同期）機能について提案し、実装方式について検討する。次に、メモリベース通信を利用して、ページ単位の効率の良いキャッシュシステムと更新型共有メモリアクセスを含む高速なユーザ通信 / ユーザ同期を可能にする非対称分散共有メモリの枠組を提案する。最後に、メモリベース通信機能の基本機能に関する実測結果を示す。

1 はじめに

マイクロエレクトロニクス技術の急速な進展に伴い、ワークステーションを高速ネットワークで結合したワークステーションクラスタ（Network of Workstations: NOW）や多数の CPU を搭載した分散メモリ型並列計算機が容易に実現可能になり、従来メインフレーム、すなわち大型 / 超大型計算機システムだけが実現可能であった応用分野に対しても適用の可能性が高まってきた。これらの分散メモリ型の並列計算環境を大型 / 超大型計算機システムとして利用するためには汎用性（マルチユーザ・マルチジョブ）の導入が不可欠である。しかしながら、汎用性導入による保護機能の実現や実資源の仮想化は、並列処理の高効率化と相容れない要素があり、並列処理の大幅な性能低下の原因となる。本稿では特殊な通信同期ハードウェアを仮定しない NOW 環境においても、高速かつ保護され仮想化されたユーザ通信 / ユーザ同期を提供するメモリベース通信機能の基本方式と実装方式を示す。次に、このメモリベース通信を利用してユーザレベルの高速高機能な共有メモリモデルを提供する非対称分散共有メモリの枠組を提案する。さらに、非対称分散共有メモリスキームに従うプログラミングモデルと実行モデルを示す。最後に、ワークステーションクラスタ上でのメモリベース通信機能の基本機能に関する実測結果を示す。

---

*The Asymmetric Distributed Shared Memory Using Memory-Based Communication Facilities.
Takashi MATSUMOTO†, Taketo KOMAARASHI‡,
Shigeru UZUHARA‡, Kei HIRAKI†
†Department of Information Science, University of Tokyo,
‡AXE Inc.

2 メモリベース通信機能

2.1 従来のユーザレベル通信の問題点

ワークステーションクラスタやユーザレベル通信用の専用ハードウェアを持たない分散メモリ型並列計算機を使用することを仮定すると、通信用インターフェースへの低レベルアクセスは保護の観点からカーネルモードで行う必要がある。また、受信インタフェースは通常受信バッファメモリ（主記憶ではなくインタフェース内に存在する場合もある）に通信内容を受け取る構成となっているが、受信バッファメモリの領域は予め設定しておく必要があり、容量は有限であることが多い。このため、受信割り込みハンドラを用意して、割り込みベースで適宜受信内容を主記憶上に退避する必要がある。このような計算機環境において従来型オペレーティングシステムの上でメッセージパッシング方式の通信を実装した際の問題点は以下の通りである。

- 高オーバヘッド
  - ユーザ / カーネル切替コスト
  - コンテクスト切替コスト
  - キュー操作オーバヘッド
  - バッファ操作（コピー等）オーバヘッド
  - ノード間コネクション管理オーバヘッド
- マルチキャスト通信のオーバヘッド
- Acknowledge(Ack) 回収のオーバヘッド
- 遠隔実行による不公平性
- 遠隔実行によるキャッシュ / TLB ポリューション

2.2 問題点ごとの分析と対策

1. 各種モード切替コスト

単にユーザ / カーネルのモード切替トラップや外部割込ハンドラへの反応時間だけで見れば、高性能マイクロプロセッサのオーバヘッドは数クロック ~ 数十クロック程度である。従来型のオペレーティングシステムでは余分なチェックや余分な再スケジューリングでモード切替コストを大幅に増大させている。対策としては以下のような実装を行う。ユーザレベルで使用するノード間通信同期用システムコールや割込ハンドラは、 OS の他の I/O 関係のシステムコールや割込ハンドラと分離して実装し、通信同期にとって余分なチェック等を全廃し、カーネル権限で実行されるコードおよびユーザ / カーネル切替に際して退避復旧するプロセッサコンテクストを最小限にする。最近の高性能プロセッサは異なるコンテクストのデータをキャッシュおよび TLB に混在可能なため、この通信同期用システムコールや割込ハンドラではキャッシュフラッシュや TLB フラッシュを行わず、キャッシュ /TLB のポリューションを最小限で済ます。

2. キュー / バッファ操作オーバヘッド

バッファのコピー回数やキュー操作のソフトウェアオーバヘッドを抑える最良の方法は以下のとおりである。通信パケット内に通信相手先のアドレス空間内の目的アドレスを格納しておき、 1. の実装方針に従って作られたメッセージ受信用の割込ハンドラが直接相手先のユーザ空間内のメモリ領域に通信パケット内のデータを (受信バッファ領域から) 格納する。この方式を用いれば CPU によるメッセージのコピーは最小の 1 回で済み、複数のユーザやジョブで共有されたバッファやキューを介さないので、それらの操作オーバヘッドは存在しない。プロテクションはパケットの発信元と受信先の少なくともどちらか一方でケイパビリティチェックを行うことで実現できる。

3. コネクション管理オーバヘッド

動的な通信はコネクションも動的に接続・解除されるので、多数のコネクションを同時に保持管理するためのコストの問題はあまり生じない。しかし、静的 (コンパイル時) に、通信相手や送信先アドレスやフェッチデータのアドレスが判る場合には、通信相手や転送先アドレスといったコネクションに関する静的情報を利用した方が、動的なコネクションを張るオーバヘッドが回避できるので効率が良い。並列実行に非常に適したデータパラレル (SPMD) の数値計算アプリケーションでは、論理アドレス空間を各プロセッサで共通 (ただしメモリの実体は別) にすることがコネクションのための静的情報の利用に関して非常に有効である。論理アドレス空間を共通にすることにより、遠隔メモリアクセスのためのコネクション情報はプログラムコード内に操作対象の論理アドレス (と静的に判明する論理プロセッサ) として格納可能になり、実行時には送信側で論理プロセッサと物理プロセッサ (物理ノード) の対応を取るだけで通信同期が可能である。通信同期パケットの受信側は割込ハンドラが論理アドレスを解決[1]してメモリ操作を行うだけである。

4. マルチキャスト・ **Ack** 回収

多数のノードに同一メッセージを送りたい場合に、 1-to-1 のメッセージを繰り返しすべての宛先に発行したのでは、送信インタフェースがソフト的にもハード的にボトルネックになってしまう。また、マルチキャストが完了したことを確認するために Ack を宛先の数だけ送信元が受け取ると、受信インタフェースがボトルネックになってしまう。この問題の解決には通信経路に何らかの階層構造を導入し、階層マルチキャストと Ack コンバイニング [1] をパケット受信割込ハンドラでエミュレートする必要がある。基本的に大規模なマルチキャストは階層構造を利用して行われ、 Ack のコンバイニングも階層構造を利用しない限り効率化できない。このため、 NOW においてはネットワーク内に階層的な経路を静的または準静的に見い出して、それを利用してマルチキャスト・ Ack 回収が実現される必要がある。つまり、宛先が頻繁に変わる場合は階層的な経路の算出のオーバヘッドが無視できなくなってしまう。この意味では本解決策はページ単位で共有される分散共有メモリのための update メッセージに最適である。

5. ユーザレベル遠隔実行の問題

ナイーブなメッセージ駆動や要求駆動などのパケット到着駆動による実行形態を基本実行形態として選択するには現在の高性能マルチプロセッサは処理の空間的時間的局所性を期待しすぎている。また、局所性の利用は、コンピュータの高速化にとって最も重要な事項であるから、不可避である。しかし、処理したいデータセットの大部分が他ノードにあるような場合は逆に通信によってコントロールを移動した方が局所性が抽出できる。この場合においても、現在実行中のジョブから CPU を奪って実行したのでは、現在実行中のジョブの局所性の利用を妨げ、スケジューリングの公平性からも望ましくない。解決策として以下の方法を採用する。遠隔実行メッセージ到着時にすぐに内容を実行せずに、ユーザレベルのランキューにパケットとして運ばれてきた遠隔実行スレッド (エントリポイントと引数) を直接格納することで、キャッシュポリューションや不公平性を回避する。このユーザレベルランキューの所有者であるジョブが実プロセッサにスケジューリングされている時しかこのキューからスレッドが起動されることはない。このため関係のないジョブの CPU 時間の消費はユーザレベルランキューへの登録操作 (データ量が多い場合は実際のデータ移動は DMA 転送) のみであり最小限である。さ

[1] 解決と言っても通常 TLB を多くとも 1 本セットするだけである。

らに、このスレッド起動はアドレス空間がすでに切り替わっており、ユーザレベルで実現されるので、極めて低コストである。

2.3 全体的な基本方針

上記の問題個別の対策全体から、カーネルレベルの処理を保護や公平性の制約が許す範囲で最小にして、共有メモリモデルに基づく高機能遠隔メモリアクセス（メモリベース通信（同期）機能）をインプリメントすることが共通の解決策となっていることが判る。つまり、メモリベース通信機能を Memory-Based Processor（MBP）[1]（ユーザレベル遠隔メモリ操作専用ハードウェア）なしに可能な限り低オーバヘッドのソフトウェアにより実装することが高速かつ仮想化され保護されたユーザレベル通信を実現する方法である。MBP がハードウェア的に行っていた処理は、ユーザレベルの送信コードとカーネルレベルの送信用トラップルーチンとカーネルレベルの受信割り込みルーチンで代替される。MBP においてもこのソフトウェアメモリベース通信においても、基本的なアイデアはユーザレベル通信同期の仮想化と保護を、論理アドレス[2]（つまり共有アドレス、MBP ではネットアドレス）を仲介して、メモリの仮想化と保護の問題に置き換えているという点にある。この置き換えを採用しているため、通信対象のタスク（アドレス空間の切替え単位）を通信パケット内に指定することにより、通信先のノードで通信対象のタスクが CPU にスケジュールされているかどうかにかかわらずノード間の軽い通信が可能である[3]。さらに、個別対策でも述べたように、MBP 用に開発された階層マルチキャスト、Ack コンバイニング、Memory-Based Signal といった各種機構がソフトウェアメモリベース通信の枠組でも利用できる。

ユーザレベル通信用のトラップおよび割り込みルーチンはブロックされない仕様となっている。ノード間通信同期に関わる低レベルの各種同期情報はすべてユーザレベルのフラグに反映され、このフラグを利用して Snoopy Spin wait（**SS-wait**）[2] と呼ばれる資源割当状況および同期達成状況によるスケジュールオプションつきのスピンウェイトによって同期が行われる。

遠隔メモリリードや各種メモリベース同期機能は要求元からの遠隔メモリアクセス要求パケットの送信、受信時に割込ハンドラ内で目的クラスタ（ノード）内の目的タスクの対象アドレスの操作（同期処理の種類によっては不可分操作）および要求元への返り値（返り値）の返送、要求元での受信割込ハンドラによる返り値の返り値アドレスへの書き込み（Ack の場合はカウントアップやカウントダウン）で実現される。

2.4 プロテクション方式

個別問題対策の 3. において述べたように、静的な情報を活用してオーバヘッドを極力削減するため、同一ジョブ（我々の用語ではプロセス）内の通信同期は自由にする。パケット内には通信相手先のプロセス識別子と改竄不可能で送信トラップルーチンにおいて付加される送信元のプロセス識別子が含まれている。パケット受信時に割り込みルーチンがこれらの識別子を比較して、同一プロセス内へのメモリベース通信である場合は受信ノードでのメモリアクセスを許可する。

ジョブ（プロセス）間のメモリベース通信に関しては、アクセス許可キーによるケイパビリティチェックによって保護の下でプロセス間メモリベース通信を可能にする。アクセス許可キーの発行はアクセス権申請のメッセージの内容を対象プロセスのユーザレベルのアクセス権チェックルーチンが検査することによって行われる。アクセス権申請メッセージはメモリベース通信の中では特殊な通信で有効な格納先論理アドレスをパケット内に持たず、相手先ノードのプロセス識別子とタスク識別子（プロセスが複数のアドレス空間をノード内に持つ場合）のみが受信先を決定するのに使用される。指定されたタスクのアクセス権チェックルーチンおよびチェックルーチン用キューがカーネルに登録されている場合のみ、受信割り込みルーチンはメッセージを登録されたキューに格納し、そのタスクの最高優先度のユーザレベルランキューにアクセス権チェックルーチンを登録する。そして、メッセージの内容がアクセス許可キーの発行条件を満たしているとチェックルーチンが判断すると、チェックルーチンが許可キーを申請元にメモリベース通信による遠隔書き込み（書き込みアドレスおよび申請元のアクセス許可キーは申請メッセージに含まれる）で発送する。不正なアクセス権申請メッセージの発行を防ぐために、アクセス権申請メッセージには申請元のアクセス許可キーのみではなく申請元の実行停止許可キーがカーネルによって付加されており、実行停止許可キーを利用したプロセスの実行停止要求メッセージをシステムはサポートしている。これにより、受信割り込みルーチンもしくはユーザレベルのアクセス権チェックルーチンは不正な申請メッセージを送り付けたプロセスを停止させることができる。受信割り込みルーチンの負荷を増やさないため、アクセス可能な範囲をコネクションごとに指定する機能は実装しない。アクセス可能範囲を相手のプロセスごとに限定したい場合は、アクセスを許可するメモリ範囲だけを含む別タスク（アドレス空間）を生成して、そこを介してメモリベース通信を行う。

[2] 厳密に述べると、ソフトウェアによるメモリベース通信では、共有アドレスを構成するのは通信先の論理アドレスのみではなく、相手先のアドレス空間を指定するためのノードの指定とタスクの指定も含まれる。

[3] タスクを他ノードにマイグレーションする場合のみ、そのタスクと通信する可能性のあるノード全体と同期を取り、ネットワーク中の該当タスクを宛先とするパケットもすべて到着する必要がある。

2.5 フロー制御と Ack 管理

Ether や FastEther や ATM のような汎用通信インタフェースを使用する場合、ハードウェアレベルでパケットが消失する可能性がある。これは純粋にハードウェアレベルの通信方式の問題であるが、現在の大多数の汎用通信インタフェースはパケット消失の危険性がある。このような通信ハードウェアを使用するメモリベース通信では、パケットに送信ノード毎にシリアルナンバーをつけて管理している。ナンバーに欠番があることが受信側で検出された場合は、欠番のパケットの再送を送信元に要求する。送信元ではパケットを受信先への到着が確定するまで保存しておく必要がある。到着の確定には次に述べる Ack が使用される。

パケット消失に備えて過去に送信したパケットを保管しているバッファの解放、メッセージ間の到着順序の保証および同期、FIFO 性のないネットワークでの FIFO 性の保証等のために Ack は不可欠である。超高速ネットワークが使用可能な場合は Ack[4]をメッセージごとに返送しても構わない。しかし、10Mbps や 100Mbps 程度のネットワークであれば極力ネットワーク上のトラフィックを削減したい。そこで、パケット消失の検出に用いている送信ノード毎のシリアルナンバーを利用して Ack の回収頻度を削減する。資源解放や順序保証のために先行する通信が終了したことを確定しなければならない場合に、確定させようとするノードから各宛先ノード毎に現在のシリアルナンバーを持った Ack チェック用のメッセージを発行する。このメッセージを受け取ったノードはそのシリアルナンバーまでメッセージを受け取っていれば Ack を返答し、受け取っていなければ Nack を返答して欠落したメッセージの再送を要求する。さらに、このチェック用メッセージおよび Ack (Nack) の消失にはタイムアウトを利用して対応する。

2.6 マルチキャスト方式

個別対策の 4. で述べたように、マルチキャスト経路に階層構造を導入し tree 状にマルチキャストを行う方式を実装する。中継ノード状の受信割り込みルーチンがノード内の対象メモリ操作を行うだけでなく、自分の子ノードを宛先としたパケットを生成してパケットを転送する。子ノードが複数ある場合は、中継ノードでパケットが増殖し、各中継ノードの並列動作により対数オーダの時間でマルチキャストが可能になる。マルチキャストの経路や各ノードでの操作対象アドレスを発行元がすべて指示する方式ではパケットサイズが大きくなりすぎる。このため、中継ノードに子ノードに関する情報が分散配置されている必要がある。この中継情報を簡略化するために操作対象の論理アドレスのページ単位に、この情報を保持することにする。すべてのノードで同一プロセスかつ同一論理アドレスへデータを書き込む update メッセージの場合は、論理アドレスの変換等は不要であるため、ページ単位に子ノードの物理ノードアドレス (ether-net アドレス等) が求められればよい。

マルチキャストの Ack 回収は前小節の Ack 管理方式と Ack コンバイニングを組み合わせて効率良く行う。

2.7 他の高速ユーザレベル通信との比較

通常の通信ハードウェアの制御レジスタをユーザ空間にメモリマップして高速のユーザレベル通信を行うものに、Fast Message[3] や PUMA-III[4] があるが、仮想化や保護を無視しているのでシステムを汎用計算機として使用することができない。

論理的な通信先ノードから物理的な通信先ノードへのマッピングをハードウェアで行い、そのマッピングをジョブ単位に切替え、受信側ではユーザレベルに公開されたジョブ毎の受信バッファを変更することで、ジョブの時分割実行に対応するシステムに CM-5[5] がある。この方式では、CPU に割り当てられているジョブへのメッセージ以外は受け取ることができないため、ジョブ切替時にネットワーク上の関連メッセージを一斉に退避する (All Fall Down) 必要がある。さらに、単一受信バッファ方式なので、ユーザアプリケーションがメッセージを利用する時にキュー操作ならびにコピーのオーバヘッドがかかる[5]。

Active Message (AM)[6] はもともと単一ジョブがシステム全体で動いているような環境で考えられた保護や仮想化の概念のない軽い通信方式であるが、最近 AM の考案者の一人からワークステーションクラスタに対応した Sparcstation Active Messages (SSAM)[7] が提案されている。AM ではユーザの用意したユーザレベルの受信ルーチンを直接起動することにより、オーバヘッドを最小限にしている。しかし、ワークステーションクラスタでは通信対象のジョブが受信時にスケジュールされているとは限らない。この問題を解決するために SSAM では一旦メッセージをメモリにバッファリングして、そのバッファをジョブから見えるようにメモリマップを変更し、ジョブが起動されているときにユーザの受信ルーチンを起動させる。遠隔書き込みに関して、SSAM は本稿で述べたメモリベース通信より、一回バッファリングの回数が増加し、バッファエリアの確保やバッファのメモリマップ変更操作等のオーバヘッドが増え、定性的に明らかに性能が劣る。何らかの付加的なユーザレベルの処理が受信ルーチンで行われることが有利な場合であっても、メモリベース通信のユーザレベルランキューに直接

[4] 受信割り込みルーチンから直接リプライのあるメッセージはリプライを Ack と見做す。

[5] この CM-5 の方式は純粋なソフトウェアによる高速方式とは言い難く、メモリベース通信も論理アドレスを物理アドレスに変換してパケットの内容をメモリに格納するハードウェアが存在すれば遠隔メモリ書き込みにおいて CPU の処理オーバヘッドは存在しない。

コンテクストを登録する方式の方が、起動時にユーザレベルで起動可能であり定性的にSSAMより優れている。

3 非対称分散共有メモリ

前節では、ユーザレベル通信専用ハードウェアを仮定しない高速かつ仮想化され保護されたユーザレベル通信同期方式として、ソフトウェアによるメモリベース通信(同期)機能を提案し解説した。このメモリベース通信をユーザプログラムで直接使って並列プログラムを記述することが可能である。メモリベース通信はある種の共有メモリモデルを仮定しており、共有アドレス(論理的な相手先ノードおよび相手先ノード内の論理アドレス)を静的に決定できればできるだけ効率が向上する。しかし、メモリベース通信を直接ユーザが使用するだけでは、遠隔メモリのキャッシュ機構がシステム側に用意されていないため、アプリケーションの効率の良い並列実行のためにはソフトウェアコントロールキャッシュとしてのキャッシュの導入が必須である。しかしこの場合、キャッシュのヒット判定、キャッシュ領域のメモリ管理、同一性の維持等をすべてユーザの責任で行う必要がある。キャッシュの明示的なヒット判定が読み出し時にも必要となり、このことは大きなオーバヘッドとなる。また、ソフトウェア分散共有メモリでページ単位でキャッシュ管理を行っている場合、主記憶の不足時にリプレース対象に次回のCPU割当まで間があるジョブのキャッシュコピーを選ぶことにより、実際に必要となった時点でネットワークを介してコピーを作成することができる。この方式は二次記憶としてディスクをアクセスするより圧倒的に有利である[8]。ユーザレベルでキャッシュ領域を管理したのでは、こういったメモリリプレース時の最適化戦略も非常に適用しにくい。

これらの理由から効率の良いソフトウェア分散共有メモリ(ソフトウェアDSM)の必要性を認識し、我々はメモリベース通信機能を使った新しいソフトウェアDSMの枠組である非対称分散共有メモリ(Asymmetric Distributed Shared Memory: ADSM)[9]を考案した。遠隔メモリ書き込みを伴う可能性がある場合に、従来のソフトウェアDSM[10]は単なるstore命令をコンパイラが用意し、ページの書き込みトラップ時にコンシステンシ維持のためのコードを実行していた。それに対して、ADSMではADSM用コンパイラが書き込み対象のデータの従うべき分散共有メモリプロトコルに適合した一連の命令(コンシステンシ維持コードを含む)を実行コードとして埋め込む。遠隔メモリ読み出しの可能性に関しては、従来のソフトウェアDSMと同様に、load命令を用意しページトラップでキャッシュミスを検出して対処する。ただし、ページトラップで起動されるページ読み出しルーチンは遠隔書き込みとして埋め込まれた一連の命令と対応している必要がある。例えば、updateプロトコルを採用したADSMの場合、遠隔書き込みに対してメモリベース通信のホーム[6]経由マルチキャストを利用したデータ送信コードが埋め込まれる、loadのページミスによるトラップルーチンではホームから最新のページ内容を読み出せば良い。遠隔メモリの読み出しと書き込みの取り扱いが異なる点がADSMの最大の特徴であり、名前の由来となっている。

遠隔メモリ書き込みのためのstore命令を一連の命令列で置き換えてしまうことにより、ADSMの自由度は非常に向上しており、様々なメモリモデルやコンシステンシプロトコル(更新型を含む)をサポート可能であり、通信量および通信回数を削減するための通信の動的コンバイニングもコードとして埋め込むことができる。さまざまなプロトコルが実現可能なことから、ADSMではページ単位のプロトコル切替[11]による最適化をサポートすることが可能である。ページ単位のソフトウェアDSMは従来はfalse sharingの問題が非常に深刻であったが、近年Release Consistency(RC)モデル[12]に基づきfalse sharing問題を緩和するソフトウェアDSM[13, 14]が考案されており、ADSMはこれらをstore命令をコンシステンシ保持動作と通信のための命令シーケンスに置き換えることで、特殊なハードウェアを仮定せずに効率良くサポートすることができる。

ADSMはノンブロッキングかつスプリットフェーズのメモリベース通信を利用しているため、一つのCPUが行った遠隔メモリアクセスに関して何ら順序制御を行っていない。メモリアクセス順序の保証には、メモリベース通信のAck回収を利用したメモリバリアもしくはエラスティックメモリバリア[15]を使用する。これらのメモリバリアはノンブロッキングの専用システムコールであり、先行する遠隔アクセスが終了しているかどうかをユーザが指定したフラグに返答する。未終了を返答した場合は終了時にそのフラグを変更する。ユーザはこのフラグを利用したSS-wait等で同期を取る。

上記から判るように、ADSMはコンパイラによる最適化を前提としたソフトウェアDSMである。RCモデルに基づきfalse sharing問題を緩和するハードウェアDSMおよびソフトウェアDSMも共有書き込みがコード上は単一のstore命令で済むとしても、RCモデルに基づいたコードを生成する特殊な最適化コンパイラが必要であることに変わりはない。また、更新型および無効化型の双方をサポートするハードウェアDSM[1]は理想ではあるが、多くのコンシステンシ管理用のハードウェアを必要とするため、NOW環境では実現不可能である。

4 プログラミングモデル

メモリベース通信およびADSMはNOWおよび分散メモリ型並列計算機に一般に適用可能なソフトウェ

[6]順次性と単一性の保持を要求するプロトコルでページ毎に一意に決められているコンシステンシ管理の中心となるノード。

ア機構である。現在、我々はこれらの機構のテストベッドとして汎用超並列オペレーティングシステム SSS–CORE [16, 9] を研究開発中である。本節からは SSS–CORE における ADSM に基づくプログラミングモデルおよび実行モデルを例に用いて、より具体的な議論を行う。まず、SSS–CORE 上の ADSM に基づくユーザのプログラミングモデルを述べる。SSS–CORE では並列実行コードの生成が比較的容易な以下の二つのプログラミングモデルを推奨モデルとする。

並列性の抽出が容易な (もしくは並列性を明示する) HPF のような SPMD タイプを推奨プログラミングモデルの一つとして採用する。このプログラミングモデルでは大域変数の宣言に特徴があり、大域変数ごとに ADSM の各種プロトコルとホームノード (論理プロセッサ番号で指定) を宣言する[7]。最適化コンパイラは同一ノードの同一プロトコルの変数をページ単位でまとめ、大域変数への書き込みではプロトコルとノードに対応したコードシーケンスを生成する[8]。また、静的に読み切れる場合は冗長なノード間通信の削除やコンバイニングを行う。また、実行時の動的なコンバイニングコードをユーザレベルで挿入する[9]。

SSS–CORE は NUMA 環境 (NOW 版 SSS–CORE では、ADSM により NOW を NUMA として扱う) に UMA 型のクラスタ (例えば、密結合マルチプロセッサ型のワークステーション) が含まれることを許している。SSS–CORE が推奨するもう一つのプログラミングモデルはクラスタ単位のマルチスレッドプログラミングとクラスタに跨る大域配列、同期変数、通信変数による共有メモリアクセスを許すプログラミングモデルである。このモデルを SSS–CORE では Multi-Thread Multi-Task (**MTMT**) モデルと呼ぶ。UMA 部分がない場合は単一プロセッサと付属のメモリをクラスタと見做して、その内部でマルチスレッディングがなされる。SPMD タイプとは異なり、個々のプロセッサにスケジュールされるべきスレッドごとにプログラムが記述されている。スレッド間のスケジュールは言語によって提供されるランタイムライブラリやマクロ関数によって行われる他、特別なスレッドであるユーザレベルスケジューラをユーザが書くこともできる。大域共有変数は SPMD モデルと同様にプロトコルの種類とホームの論理位置を宣言する。この MTMT モデルは従来の実行モデルに対応しており、クラスタ内はプロセッサ台数に関わらず完全に同一アドレス空間が実現されており、ユーザレベルスケジューリングによってスレッドのスケジューリングがなされる。ただし、ユーザレベルでクラスタを跨いで他のクラスタにスレッドの実行を移動させることはできない。(OS によってクラスタ単位でマイグレーションされる可能性はある。)

SSS–CORE には以下の属性を持つメモリ領域 (ページ単位) を持つ[10]。ただし、ADSM としての機能を持つページの書き込みは invalidate ページを除いて、前述のメモリベース通信機能を用いた一連のコードシーケンスで達成される。

local ページ SPMD モデルにおいてこの属性がデフォルトであり、クラスタ外、クラスタ内を問わず、論理プロセッサごとに同一の論理アドレスに対して別の物理領域が割り当てられ、コンシステンシの維持はなされない。MTMT モデルでは存在しない。

c_local ページ このページ上の変数は同一クラスタ内でのみ、同じ物理領域を指す。MTMT モデルにおけるローカル変数はすべてこのクラスである。

home_only ページ 読み書きを問わず遠隔メモリアクセスを要求し、コピーを他ノードに生成しない領域。ページの一部しか利用しないことが判っている時には有利。

invalidate ページ ADSM の一種。ホーム以外に同一論理アドレスのコピーページを生成する。ページ単位の無効化によって Sequencial Consistency (SC) でコンシステンシ管理がなされる。

ulrc ページ ADSM の一種でユーザコード支援による Lazy Release Consistency (LRC)[13] を実現するページ。ホーム以外に同一論理アドレスのコピーページを生成する。LRC と同様なページ単位無効化によりコンシステンシ管理がなされる。

home_update ページ ADSM の一種で Automatic Update Release Consistency (AURC)[14] を専用ハードウェア不用で実現するページ。ホーム以外に同一論理アドレスのコピーページを生成する。書き込みはホームページへの遠隔メモリ書き込みとローカルコピーへの書き込みを行い、AURC と同様な手法によりコンシステンシ管理がなされる。

update ページ ADSM の一種でホーム経由の update プロトコルを専用ハードウェア不用で実現するページ。ホーム以外に同一論理アドレスのコピーページを生成する。メモリベース通信の update 機能が書き込みで使用される。RC ベースの同期がアプリケーションコードに挿入される。

update_drct ページ ADSM の一種でホームを経由しない update プロトコルを専用ハードウェア不用で実現するページ。ホーム以外に同一論理アドレスのコピーページをプログラムロード時または大域的なメモリ割り付け時に用意する。このコピーページは対応するホームページが消去されない限り消去されない。メモリベース通信の update_drct 機能が書き込みで使用される。RC ベースの同期がアプリケーションコードに挿入される。

[7] 宣言がないと言語のデフォルトのプロトコルが用いられ、デフォルトの領域分散ルールでホームノードが決まる。

[8] 間接参照やポインタで静的にホームノードが不明の場合は論理アドレスからホームノードを解析するコードも書き込みシーケンス内に埋め込む。

[9] 現在のワークステーションでは細粒度通信に耐える通信能力はない。OS レベルでもコンバイニングのサポートは行う予定。

[10] ADSM では書き込み時のユーザコードによって列挙した以外の自由なプロトコルを実現することができる。

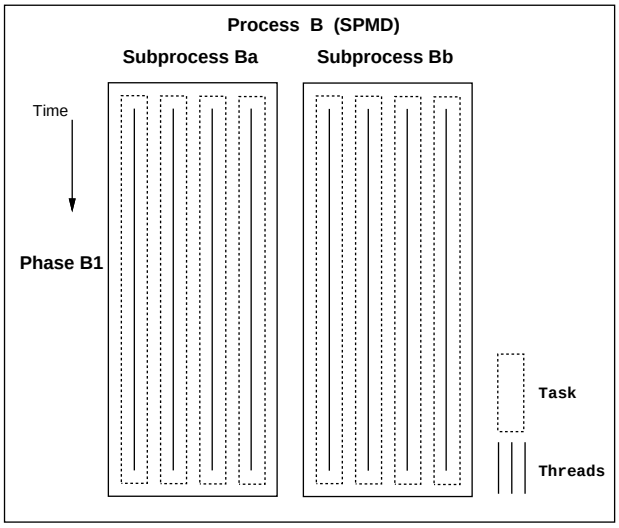

図 1: SPMD 型プロセス実行モデル

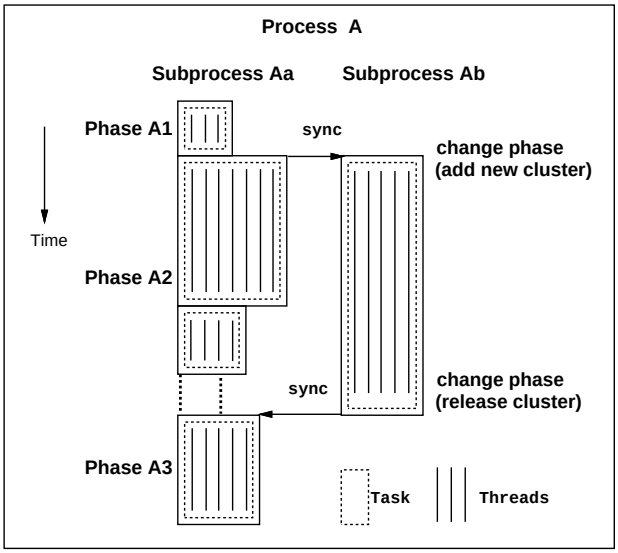

図 2: MTMT 型プロセス実行モデル

5 実行モデルとアドレスマッピング

一台のプロセッサに割り当てられるプログラムの実行の軌跡（命令流）のことをスレッドと呼ぶ。複数のスレッドで 1 つのプロセスを構成し、プロセスの内で同一クラスタに属するスレッドがサブプロセスを構成する。さらに、プロセス内の論理的にも物理的にも完全にメモリ空間を共有するスレッドがタスク [11]を構成する。タスクはサブプロセスと一致する場合もあり、複数のタスクが一つのサブプロセスに包含される場合もある。しかし、タスクが複数のサブプロセス（つまりクラスタ）に跨って存在することはない。プロセスの識別子（プロセス ID）はシステムで一意であり、プロセス ID が一致するサブプロセス / タスク / スレッドは同一プロセスに属している。メモリベース通信は相手の論理空間アドレスで通信がなされるため、タスク ID で通信相手や送信元が記述される。SPMD 型のプログラムのナイーブな実行モデルでは 1 タスクに 1 プロセッサが割り当てられ一時点では 1 タスクにおいて 1 スレッドが実行される。従って、マルチプロセッサ構成のクラスタを持つ場合はサブプロセスに複数のタスクが包含される。（図 1）。MTMT 型のプログラムではタスクはクラスタ内の複数のプロセッサの割当を同時に受けることができ、複数のスレッドが同時にスケジュールされ得る（図 2）[12]。

異なるサブプロセスに属するタスク / スレッドであっても通信や同期の必要なメモリ領域には少なくとも ADSM の意味の論理的共有メモリが実現される。つまり、同一プロセスであればグローバル共有変数はプログラミングモデル上、同一アドレス空間に属し共有されている。ただし、クラスタやタスクが異なるスレッドのプライベート領域はたとえ実行コード上の論理アドレスが同一であったとしても共有されない。

プロセスのプロセッサ希望割り当て台数や実メモリ割り当て量はアプリケーションプログラム内でも時時刻刻と変化しており、特にプログラムの繰り返し構造が変更された時点で大幅に変化することがある。このため、サブプロセスにフェーズという概念を導入し、サブプロセスが複数のフェーズに時分割されると考え、フェーズ変更時にはスレッドの種類やサブプロセスレベルの資源要求数を変更可能にする。プロセス全体として資源要求の大きさが変更される場合はサブプロセス間で同期を取り、プロセス全体の要求としてカーネルに要求が出される。カーネルはプロセス（サブプロセス）のロード時とフェーズ変更時（実際の変更よりも前に要求は出せる）にユーザからのカーネルレベルスケジューリングのための情報を入手する。

6 メモリベース通信の初期評価

メモリベース通信の基本となる遠隔メモリ書き込みを使った細粒度の通信能力に関して、現在開発中の NOW 版 SSS–CORE 上にメモリベース通信の一部を実装し初期評価を行った。評価に使用した NOW 環境は Axil 320 model8.1.2 (Sun SS20 互換機, 85MHz SuperSPARC CPU × 2) 6 台を 10BaseT のハブで EtherNet 接続したものである。

基本性能値である送信時 / 受信時のオーバヘッドは表 1に示す通りである。時間は 0.5μsec 単位の時計で計測した。 表 1に示されるオーバヘッドはすでにかなり低い値であるが、送受信ルーチン完成から日が浅いため、まだ最適化の余地はあると考えられる。なお、この計

[11]メモリ空間（ページテーブル）の割り当て単位であり、従来の OS ではこのタスクがプロセスと呼ばれることが多い。過去の SSS–CORE の文献ではサブプロセスがメモリ空間の割り当て単位を兼ねていた。

[12]従来の SSS–CORE の関連文献では MTMT モデルに相当する実行モデルしか示していなかった。しかし、SPMD 型プログラムのローカル変数を効率良くサポートするためにタスクの概念を導入し、SPMD 型の実行モデルを新設した。

表 1: 遠隔メモリ書き込みの送受信オーバヘッド

転送サイズ (byte)	64	32	16	8	4	2	1
送信コスト (μsec)	11	10	9	9	8	6.5	6.5
受信コスト (μsec)	15	10.5	8.5	7.5	7	7.5	7

測に使った受信ではアクセスコストの高いフレームバッファに直接書き込みを行っている。 EtherNet では最小パケットサイズが 76byte 相当であるため、 10Mbps の 10BaseT ではどのケースでも明らかに転送がボトルネックになっている。

7 おわりに

本稿では特殊な通信同期ハードウェアを仮定しない NOW 環境および分散メモリ型並列計算機において、高速かつ仮想化され保護されたユーザレベル通信同期であるソフトウェアによるメモリベース通信 (同期) 機能について提案し、実装方式について検討した。次にメモリベース通信を利用して、ページ単位の効率の良いキャッシュシステムと更新型共有メモリアクセスを含む高速なユーザ通信 / ユーザ同期を可能にする非対称分散共有メモリ (**ADSM**) の枠組を提案した。 ADSM スキームでは共有メモリの読み出しは従来の共有仮想メモリの方式に従い、書き込みと同期はユーザコードに一連の命令シーケンスを静的に挿入することで実現される。最後に、メモリベース通信機能の基本機能に関する実測結果を示した。実測によると 32byte 以下の細粒度遠隔書き込みであれば、送受信共に数 μsec 程度のオーバヘッドで 1 回の通信が可能である。このことは 100BaseT や高速光リンク等の速い通信ネットワークを使用した場合に、ユーザレベル通信同期専用ハードウェアがなくても高速かつ保護され仮想化されたユーザレベル通信同期が実現できる可能性を示唆している。また。効率の良いメモリベース通信の実現は柔軟かつ高効率な ADSM の実現にも継っていく。

謝辞

本研究は情報処理振興事業協会「独創的情報技術育成事業」の一環として行われたものである。

参考文献

[1] 松本 尚, 平木 敬: Memory-Based Processor による分散共有メモリ. 並列処理シンポジウム JSPP '93 論文集, pp.245–252 (May 1993).

[2] 松本 尚: マルチプロセッサ上の同期機構とプロセッサスケジューリングに関する考察. 計算機アーキテクチャ研究会報告 No.79-1, 情報処理学会, pp.1-8 (November 1989).

[3] S. Pakin, M. Lauria, and A. Chien: High Performance Messaging on Workstations: Illinois Fast Message (FM) for Myrinet. *Proc. Supercomputing'95* (Decenber 1995).

[4] 小林 伸治, 陣崎 明: PUMA-III における各種メッセージプロトコルの実装と評価. 第 53 回情報処理学会全国大会講演論文集, 第 1 分冊, pp.35–36 (September 1996).

[5] *Connection Machine CM-5, Technical Summary.* Thinking Machines Corporation, Cambridge, Mass. (November 1992).

[6] T. von Eicken, D. E. Culler et al.: Active Messages: A Mechanism for Integrated Communication and Computation. *Proc. 19th Int. Symp. on Computer Architecture*, pp.256-266 (May 1992).

[7] T. von Eicken, A. Basu, and V. Buch: Low-Latency Communication Over ATM Networks Using Active Messages. *IEEE Micro*, pp.46-53 (February 1995).

[8] 信国 陽二郎, 松本 尚, 平木 敬: 汎用並列 OS SSS–CORE におけるカーネルスケジューリング方式 — 詳細確率モデルによる性能評価 —. 情処研報 OS, SWoPP96 (August 1996).

[9] 松本 尚, 平木 敬: 汎用超並列オペレーティングシステム: SSS–CORE — ワークステーションクラスタにおける実現 —. 情報処理学会研究報告 96–OS–73, 情報処理学会, Vol.96, No.79, pp.115–120 (August 1996).

[10] K. Li: IVY: A Shared Virtual Memory System for Parallel Computing. *Proc. 1988 Int. Conf. on Parallel Processing*, St. Charls, IL, pp.94-101 (August 1988).

[11] 松本 尚: 細粒度並列実行支援機構, 計算機アーキテクチャ研究会報告 No.77-12, 情報処理学会, pp.91-98 (July 1989).

[12] K. Gharachorloo, D. Lenoski, J. Laudon, P. Gibbons, A. Gupta, and J. Hennessy: Memory Consistency and Event Ordering in Scalable Shared-Memory multiprocessors. *Proc. the 17th Int. Symp. on Computer Architecture*, pp.15-26 (May 1990).

[13] P. Keleher, A. L. Cox, and W. Zwaenepoel: Lazy Consistency for Software Distributed Shared Memory. *Proc. the 19th Int. Symp. on Computer Architecture*, pp.13-21 (May 1992).

[14] L. Iftode, C. Dubnicki, E. W. Felton and K. Li: Improving Release-Consistent Shared Virtual Memory using Automatic Update. *Proc. the 2nd Int. Symp. on High-Performance Computer Architecture*, pp.14-25 (February 1996).

[15] 松本 尚, 平木 敬: Elastic Memory Consistency Models. 第 49 回情報処理学会全国大会講演論文集 (6), pp.5–6 (September 1994).

[16] 松本 尚, 平木 敬: 汎用並列オペレーティングシステム SSS–CORE の資源管理方式. 日本ソフトウェア科学会第 11 回大会論文集, pp.13–16 (October 1994).