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 —

松本 尚[†] 駒嵐 丈人[‡] 渦原 茂[‡]
竹岡 尚三[‡] 平木 敬[†]

特殊な通信同期ハードウェアを持たないワークステーションクラスタ環境および分散メモリ型マルチプロセッサ上の汎用オペレーティングシステムである SSS–CORE において、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 SSS–CORE のプログラミングモデルと実行モデルを示し、この共有メモリを実現するための SSS–CORE 上のメモリベース通信機能を提案する。最後に、ワークステーションクラスタ版 SSS–CORE の現在の実装状況を示し、SSS–CORE 上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。

A General-Purpose Massively-Parallel Operating System: SSS–CORE
— Implementation Methods for Network of Workstations —

Takashi MATSUMOTO[†], Taketo KOMAARASHI[‡],
Shigeru UZUHARA[‡], Shozo TAKEOKA[‡] and Kei HIRAKI[†]

We propose a novel memory sharing scheme "Asymmetric Distributed Shared Memory: ADSM", that realize user-level protected high-speed high-functional communications/synchronizations. The ADSM scheme is developed for the general-purpose massively-parallel operating system: SSS–CORE on Network of Workstations (NOW) or distibuted-memory multiprocessors. We describe two programming models and two execution models suitable for ADSM on the SSS–CORE. We also introduce Memory-Based Communication Facilities of the SSS–CORE which enable a high-speed impremention of ADSM. Next, we describe the implementation status of the current SSS–CORE on NOW. Finally, we show actual measurements of exection times and communication overheads in fine-grained applications on the SSS–CORE.

1. はじめに

汎用超並列オペレーティングシステム**SSS–CORE** [1]（図1参照）は並列アプリケーションと協調動作することで、効率を極力落すことなくマルチユーザ／マルチジョブの汎用環境を実現する分散メモリ型並列計算機およびワークステーションクラスタ環境（NOW: Network of Workstations）を対象とした汎用オペレーティングシステム（汎用OS）である。SSS–CORE はシステムの資源管理に階層性を導入して資源管理の効率化を行うことにより、スケーラビリティつまり超並列超分散計算環境に対応している。SSS–CORE は共有メモリモデルに基づくプログラミングモデルをサポートする。また、並列アプリケーションのマイグレーション能力を持つ。

SSS–CORE では Snoopy Spin wait（**SS-wait**）[2] と呼ばれる資源割当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を同期の基本としている。SS-wait をユーザレベルで効率良く実現するためには OS 保護の下で軽く行えるユーザレベルの更新型分散共有メモリアクセスが必要である。ユーザの並列アプリケーションにおいても、更新型分散共有メモリアクセスが可能であれば、効率良く実現できるアルゴリズムが多数存在する。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、更新型共有メモリアクセスを含む高速なユーザ通信／ユーザ同期を提供する**非対称分散共有メモリ（Asymmetric Distributed Shared Memory: ADSM）**の枠組を示す。さらに、ADSM スキームの SSS–CORE のプログラミングモデルと実行モデルを示す。次に、ADSM を実現するためのメモリベース通信機能を簡単に述べる。最後に、NOW 版 SSS–CORE の現在の実装状況と SSS–CORE 上のメモ

† 東京大学 大学院理学系研究科 情報科学専攻, Department of Information Science, University of Tokyo
†† 有限会社 アックス, AXE Inc.

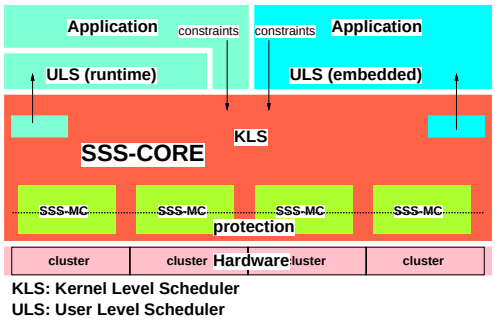

図1　SSS–CORE の構成

リベース通信機能の基本機能に関する実測結果を示す。

2. 非対称分散共有メモリ

複数のノードに跨る大域的なデータを自由に相互参照可能にするためには、大域的な ID（認識番号）をデータに持たせる必要がある。そして、どの ID のデータがどこにあるかという位置情報をサーチが容易な形式で管理する必要がある。この目的に最も適した ID はノード間で共有された論理アドレスに他ならない。現在の多くの高性能マイクロプロセッサはメモリページ管理機構を持っているため、アプリケーションごとに自由な論理アドレス空間を設定することができる。さらに、論理アドレスが 64bit 構成であれば、論理アドレス内にノード ID や各種属性を埋め込んでいても、十分な台数のワークステーションのメモリ全体をアドレッシングすることができる。この原理に最初に着目し応用したのが共有仮想メモリを名乗った IVY [3] である。ただし、IVY はページ単位にキャッシングを行う Sequential Consistency（SC）メモリアクセス順序モデル に従う無効化型の分散共有メモリ（DSM）であったため、false sharing に非常に弱い。

その後、ハードウェア DSM の方から提案された Release Consistency [4] モデルを応用した IVY タイプのソフトウェア DSM が開発され、特別な共有メモリ用ハードウェアなしに false sharing に対応している。特に、Lazy Release Consistency（LRC）[5] プロトコルは、データが必要となる（acquire）時点までコピーページへの書き込みの反映を遅延し、必要な差分だけを必要とするノードにまとめて転送する。この方式により LRC はソフトウェア DSM のノード間転送量と転送回数を大幅に削減している。一方、IVY のグループは LRC を基本として、簡単なノード間ユーザレベルデータ転送ハードウェアを仮定することで、LRC のソフトウェアオーバヘッドを大幅に削減する Automatic Update Release Consistency（AURC）[6] プロトコルを提案している。

LRC および AURC はいずれも基本的に無効化型プロトコルであり、共有同期変数や同報通信変数に向いているとは言い難い。また、これらのプロトコルでは共有メモリ空間の書き込みをプロセッサの store 命令で行うことを死守しようとしている☆。一方、HPF などの言語では、共有メモリモデルを提供し、最適化コンパイラによって効率の良い遠隔メモリアクセスのコードシーケンスを実行コードに埋め込んでいる。これらの事実を考え合わせると、ソフトウェア DSM のようにページ単位のキャッシュを用意し、共有領域の読み出しはコピーページもしくは本体のページ（ホームページ）からプロセッサの load 命令で行うが、共有領域の書き込みに関してはコンシステンシ管理のための付加的メモリ操作や遠隔メモリ操作のコードシーケンスを実行コードに埋め込むという新しいタイプの分散共有メモリが発想される。メモリの読み出しと書き込みの取り扱いが異なる点を特徴としており、この分散共有メモリを**非対称分散共有メモリ**（ADSM）と命名する。書き込みが一連のコードシーケンスで良いため、無効化型のプロトコルに因われる必要はなく、どんなタイプのプロトコル☆☆でも実装が可能である。よって、ADSM ではページ単位のプロトコル切替 [7] による最適化をサポートする。さらに、コードシーケンスによる書き込みであるため、書き込みをコンパイル時や実行時にコンバイニングすることが可能であり、ノード間メッセージ数を削減することができる。また、RC モデル等の緩和されたメモリモデルを仮定するのであれば、同一共有アドレスへの書き込みが同一同期区間で繰り返されることが判っていれば、最後の書き込みだけを外部に通信するシーケンスに変換し、先行する同一アドレスへの書き込みは単なる store のコードに変換する最適化を行うことができる。IVY と同じ SC モデルの無効化型のプロトコルであれば、共有メモリへの書き込みコードシーケンスは単なる store 命令 1 個となり、ADSM の特殊ケースでありソフトウェア DSM と一致する。LRC は共有書き込みコードシーケンスが書き込みの履歴を管理することで実現可能であり、AURC は履歴を管理しつつ自分のコピーページへの store とホームページへのデータ更新用通信を行うことで実現できる☆☆☆。このように、ADSM の枠組の中で様々なメモリコンシステンシモデルと様々なプロトコルが使用可能となる。なお、コンシステンシ維持完了の同期に関してはメモリベース通信機能のメモリバリア機能を利用した SS-wait をアプリケーションコード内に挿入することで行われる。

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 DSM である。RC モデルに基づくハードウェア DSM およびソフトウェア DSM も共有書き込みが単一の store で済むと言っても、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

☆ ページの書き込みトラップでライトアクセスを検知して、トラップルーチンで複雑なプロトコル処理を実行する。

☆☆ もちろん読み出し時に特別なアクションを要求するような一般的でないプロトコルは実現できない。その場合は読み出しもコードシーケンスで実現する必要がある。

☆☆☆ LRC は特殊ハードウェアを仮定しないので、本来の LRC の実装同様アプリケーションコード中は store 命令のみで、ページトラップルーチンでコンシステンシ管理を行ってもよい。

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 [9] 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。

3. プログラミングモデル

低レベルのユーザ用通信同期ライブラリとして提供される後述のメモリベース通信機能を明示的にユーザが直接使ってプログラムしても構わない。しかし、基本的に最適化コンパイラによって生成されたユーザオブジェクトコードが実行されることを念頭に SSS–CORE は設計されている。 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採用する。

並列性の抽出が容易な（もしくは並列性を明示する）HPF 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☆1。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☆2。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☆3。

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☆4。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、後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。 SSS–CORE はページフォルト時のホームノードからのメモリフェッチやメモリベース通信機能をユーザに提供することにより、これらのページの効率の良い実装を支援している。

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 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 LRC 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 AURC 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。

論理アドレスとノード内の物理アドレスの alias 機能を利用して、故意にノード間のコンシステンシを無視し

☆1 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。

☆2 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。

☆3 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。

☆4 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。

た読み書きを行うことがユーザの責任下でプログラミングモデル上は許されている。

4. 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** ☆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。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☆☆。

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。

5. メモリベース通信機能

SSS–COREにおける**メモリベース通信（同期）機能** [8] とはMemory-Based Processor（MBP）[9] と共に提案された高機能分散共有メモリシステム（SMS: Strategic Memory System）上の各種メモリベース通信機能やメモリベース同期機能を、NOWや分散メモリ型並列計算機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したものに他ならない。

OSのプロテクションの下でバッファのコピー回数やソフトウェアオーバヘッドを抑える最良の方法は、メモリ管理機構と協調動作する遠隔メモリアクセス用のハードウェア（つまりMBPのような物）を使用することである。この遠隔メモリアクセス用ハードウェアを仮定しない場合は、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、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。ユーザレベルで使用するメモリベース通信同期用システムコールおよび割込ハンドラは、OSの他のI/O関係のシステムコールおよび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して、実装される。通信同期要求パケットを受信した割込ハンドラが対象タスクの論理アドレスを解決☆☆☆してメモリ操作を行う。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。SSS–COREの現在の実装ではパスワード（32bit数値）を利用したケイパビリティチェックを受信先の割込ハンドラで行っている。MBPによるSMSの大きな特徴に階層マルチキャストとAcknowledge(Ack)コンバイニングによるupdateベースのキャッシュの実現がある。SSS–COREのメモリベース通信においても階層マルチキャストやAckコンバイニングをパケット受信割込ハンドラでエミュレートすることで実現する。NOWにおいてはネットワーク内に階層的な経路を静的に見い出して、それを利用して実現される。もちろん、並列度が小さい時は単なるチェインによるupdateを行う。

また、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。

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 [2] によって同期が行われる。遠隔メモリアクセスの順序モデルには緩和されたメモリアクセスモデルを利用することで性能を向上させる。遠隔メモリアクセスの順序管理はAckベースのメモリバリアで行われるが、可能な限りAckを省略して済ませるように実装を行う。

6. NOW版 SSS–COREの開発現状

現在、NOW版SSS–COREのSun Microsystems社のSPARCstation 20またはこの互換機をEtherNetで接続した環境で動作している。SSS–COREのWS単

☆ 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。

☆☆ 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。

☆☆☆ 運悪く異なるタスクがプロセッサで走っている時に割り込みが発生した場合でも、TLBを1本程度セットするだけである。

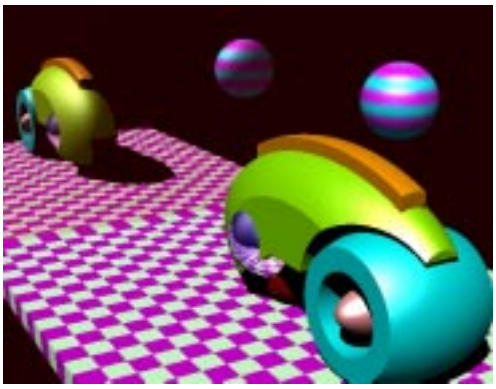

図2　レイトレ画像（実際は 256 色カラー）

体担当部分（SSS-MC: Micro Core 図 1参照）では、メモリ保護、タイムシェアリング、プロセス管理、UDP 通信、TCP/IP 通信、キー入力、画面出力、外部 UNIX ワークステーションからのファイルの読み書き／プログラムロード／実行等が実装されている。メモリベース通信機能は最もベースとなる遠隔メモリ書き込みがすでに実装されており、現在他のプリミティブを鋭意作成中である。メモリベース通信機能の実装が済み次第、ADSM の実装に着手する。ユーザのスケジューリング制約を守って資源をプロセスにスケジューリングするカーネルレベルスケジューラ（KLS）は、基本的な考え方が文献 [1] に示されており、確率モデルの精密なシミュレーションでスケジューリングポリシの詰めがなされている [10]。このシミュレーション結果を参考に、KLS の最初のバージョンが実装される予定である。

7. 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。

まず、基本性能値として、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。

次に、並列レイトレーシングプログラム（以下レイトレ）を使って、SSS–CORE 上で遠隔メモリ書き込みでフレームバッファ表示を行う実験を行った。実験に使用した 3D ソリッドモデルの完成時の絵を図 2に示す。この絵を 576 × 450 の解像度で並列計算を行った。並列計算の方法はサイクリックに N ピクセルずつを割り当て、N ピクセルのカラー値（R,G,B 各 8bit）を計算した後、 dither 変換を掛けた N ピクセル分の Nbyte データを 1 パケットとして表示ノードのフレームバッファに遠隔メモリ書き込みで直接書き込む（6 台の内 1 台は表示専用とした）。ただし、各クラスタにつき 1 台のプロセッサを使用し、各プロセッサはすべて 576 × 450 回のループを回り、ピクセルが自分の担当領域かどうか実行時に判断し、担当外であればスキップしている。時間は 100msec 単位で計測した。実験結果を表 2に示す。 本実験における 1 ピクセル当たりの計算時間は平均約 70μsec(ピクセル上の絵の複雑さによって大幅に変化) である。表 2でアスタリスクが付いている項目は EtherNet の通信競合により大幅に負荷分散が狂って実行時間が安定しない（項目にもよるが結果に± 0.3sec 程度の幅が存在する）実験項目である。表内には 3 回測定した平均が記入されている。転送サイズが 32byte 以下のパケットではヘッダーやダミー等も含めて 76 バイト分のパケットが毎回 Ether で通信されている。例えば、4byte パケットの通信では 4.92Mbyte ものパケットが送られる計算となる。これから 4byte/5 台の EtherNet の転送レートは 648 Kbyte/sec となり、ほぼ飽和状態である。他のアスタリスクの項目も同様である。なお、現在の実装の SSS–CORE は 16 回再送して送れないパケットは送信を諦めているため、アスタリスクの項目ではピクセルが何点か欠ける場合がある。また、実行時のパケットのコンバイニングをこの実験では行っていない。 1 台の計算時間が 4byte 以下になるとパケット数増加による送信コストの増加を大幅に上回る実行時間の増加が発生する。これは計算時間で通信が隠せなくなったためである。比較的大きな粒度の通信でも、台数に関してリニアスピードアップに達していないのは、各プロセッサが担当以外のピクセルに対して若干の処理を行っていることと Ether 通信の衝突に起因するものである。

次に、 SSS–CORE の軽いメモリベース通信と UNIX において軽いと言われている UDP を使ったソケット通信の性能を実アプリケーション上で比較する。このために UDP のソケットで通信するプログラムに前記レイトレを書き換えたものを用意して、表示プログラムは SunOS 4.1.3 上の X11R6 で表示し（ただし XFlush はしない）、計算プログラムは Solaris 2.4 上で動かした。使用マシン環境は SSS–CORE 用の環境の OS を交換してまったく同じにした。アプリケーションのコンパイル条件も同一のコンパイラを用い、最適化オプション（O4）も同一にした。表 3に実験結果を示す。測定は実時間で行い、 UDP 通信の送信側は送信バッファに書き込めない場合は書き込めるまで待つようにプログラムしたが、オーバヘッドを必要以上に増やさないため、それ以上の転送の保証を行わなかった。このことにより、2 台以上の実行ではどのケースでもピクセルを取りこぼす。 3 台の実行では大幅にピクセルが欠けるものが多い。 64byte 転送においては UNIX 上の UDP が勝っているが、これは浮動小数点ルーチンが異なる等のベース性能の違いである。転送の粒度が細かくなるにつれて、

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

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
1 台計算時間 (sec)	18.1	18.2	18.4	19.0	20.5	27.0	37.7
2 台計算時間 (sec)	9.3	9.4	9.5	9.7	10.5	14.1	28.2
3 台計算時間 (sec)	6.8	6.5	6.5	6.7	8.0	*14.3	*26.9
4 台計算時間 (sec)	4.9	4.9	5.0	5.2	*7.8	*13.8	*25.3
5 台計算時間 (sec)	4.0	4.0	4.1	*4.3	*7.6	*13.7	*25.5

表2　転送サイズと台数による並列レイトレ計算時間

転送サイズ (byte)	64	32	16	8	4	2	1
1 台計算時間 (sec)	17.6	18.5	19.5	22.2	29.7	41.5	68.4
2 台計算時間 (sec)	9.1	9.4	10.1	11.4	15.3	21.3	31.3
3 台計算時間 (sec)	6.6	6.6	*7.3	*8.2	*10.3	*15.2	*25.7

表3　UDP 通信による並列レイトレ計算時間

SSS–CORE の実行時間よりも長くかかるようになる。4byte 以下の細粒度転送でも 2 台以上では差が小さくなっているが、これは UDP 転送が送信再送をまったく行っていないためである（ピクセルは大幅に欠ける）。

8. 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マルチプロセッサにおいて、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（Asymmetric Distributed Shared Memory: 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。さらに、ADSM スキームに従う SSS–CORE のプログラミングモデルと実行モデルを示した。次に、ADSM を実現するための SSS–CORE 上のメモリベース通信機能を提案した。最後に、NOW 版 SSS–CORE の現在の実装状況を示し、SSS–CORE 上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した。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。通信頻度が多いアプリケーションを使った比較において、SSS–CORE は UNIX よりも良好な結果を示した。

謝　辞

本研究は情報処理振興事業会（IPA）が実施している独創的情報技術育成事業の一環として行なった。

参 考 文 献

1) 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).

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) 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).

4) 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).

5) 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).

6) 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).

7) 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).

8) 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 のメモリベース通信機能. 第 53 回情報処理学会全国大会講演論文集, 発表予定 (September 1996).

9) 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).

10) 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).