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

松本 尚　　平木 敬

東京大学 大学院理学系研究科 情報科学専攻*

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] 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。ユーザの並列アプリケーションの効率の良い実行のためには、もちろん従来 SSS–CORE で主張していたユーザ／カーネルの協調資源割当や資源管理効率化によるカーネルコストの削減も重要である。しかし、第一義的にはユーザモードにおいてノード間における通信と同期をいかに高速に実現するかが最大の鍵である。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、高速なユーザ通信／ユーザ同期を提供するメモリベース通信機能の基本方針と実装方式の概略を示す。

2　高速ユーザ通信同期実現の問題点

以下に高速ユーザ通信同期実現のための障害となってきた問題点を列挙する。

2.1　ユーザ／カーネルおよびコンテクスト切替コスト

汎用環境を実現しようとしているため高速ユーザ通信同期であっても OS によるプロテクションを廃止できない。現在の多くの分散メモリ型高並列計算機では各ノードでシングルノード用の OS を動かして、従来 OS の保護された通信同期機能を利用している。このため、非常に大きなユーザ／カーネル切替およびコンテクスト切替のオーバヘッドが通信や同期のたびに必要となる。

2.2　メッセージキュー操作のオーバヘッド

多くの分散メモリ型並列計算機や NOW では他ノードから飛んで来たメッセージパケットを少数個（通常 1 個）の通信用デバイスで受信して、とりあえずメモリ内の受信バッファ領域に格納する。ここまでの処理はハードウェアの仕様で固定されており通常回避不可能である。この後、多くのシステムでは以下のような手順となる。OS がそのメッセージをプロセス毎に仕分けしてハードウェアの専用バッファ領域から移動し、メッセージを待っていたユーザプロセスを起動状態にする。起動されたユーザプロセスはメッセージの内容を読んで処理内容を選択するために移動して、さらに最終的な処理を行うための領域に移動して、依頼された処理を行う。このようにパケットデータの移動や解読が場所を変えてくり返されて効率が悪い。

2.3　ノード間コネクション保持のオーバヘッド

前記のメッセージキュー操作のオーバヘッドを緩和するために、他ノードのユーザメモリ空間を比較的低コストで操作する遠隔メモリアクセスを分散メモリ型並列計算機や NOW でも取り入れつつある。しかし、この場合でもノード間のメモリのマッピングをユーザもしくは OS が管理保持しなくてはならないので、超並列超分散環境で通信同期するノード数および頻度が大きくなるとコストが増大する。

2.4　マルチキャスト通信のオーバヘッド

コネクション保持のコストとも関連するが、対象ノード数が増大すると 1-to-1 ベースの通信手段の繰り返しで実現される 1-to-many 通信のオーバヘッドが非常に大きい。特殊なブロードキャスト通信手段がシス

2.5　キャッシュ／TLB のポリューション

メッセージ駆動型の実行形態やデータ駆動型の実行形態や通常の高性能マイクロプロセッサを利用した並列計算機における Active Message[4] 流の実行形態では、処理が本来持つ局所性の利用が難しい。汎用環境では無関係なジョブの実行が細粒度で混じり合うので、キャッシュや TLB のポリューションの度合がより一層悪化し、性能が低下してしまう。現在の局所性の利用が大前提となっている高性能マイクロプロセッサを要素プロセッサとして使用する限り、これは大きな欠点である。

3　メモリベース通信機能の特徴

前記の問題点が認識されれば解決策は簡単であり、それは共有メモリプログラミングモデルに基づく高機能遠隔メモリアクセス、つまり**メモリベース通信（同期）機能**である。SSS–CORE におけるメモリベース通信機能とは Memory-Based Processor（MBP）[2] 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 [3] 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。以下、どのようにして前記の問題点が解決されるか簡単に説明することにより、メモリベース通信（同期）機能の特徴を述べる。

1. 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。そこで付随するオーバヘッドを最小限に抑えるために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して、実装される。この通信同期用システムコールや割込ハンドラではアドレス空間の切替を行わない。
2. 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、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。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。
3. 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、コネクション保持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

納可能になり、実行時には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。通信同期要求パケットの受信割込ハンドラが対象論理プロセッサの論理アドレスを解決[1]してメモリ操作を行う。

4. MBP による SMS の大きな特徴に階層マルチキャストと 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。この update ベースのキャッシュシステムは上記の SPMD プログラムの例と同様に論理アドレス空間を各ノードで共通に取り、論理アドレスが同じキャッシュ用のページを各ノードに用意して、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できる。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ので、この update キャッシュの実現方式は NOW であれ、分散メモリ並列計算機であれ、変わりはない。そこで、 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。
5. 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の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。このため SSS–CORE では、 SMS の Memory-Based Signal と同様に、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実行スレッド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。

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（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[5] 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。また、実装上の問題としては、 Ether や FastEther のようにデバイスレベルでパケットが消失する可能性がある通信ハードウェアに対応するために、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。紙面に限りがあるため、メモリベース通信機能およびその実装方式の詳細については別稿で報告する予定である。

4 メモリベース通信機能に適したプログラミングモデル

前出の update 系書き込みと遠隔メモリ書き込みアクセスは、プログラミングモデル上はすべてのノードにおいて単なるメモリアクセスとして記述されるが、 NOW 上の SSS–CORE における実行コードでは、ングモデルを採用する必要がある。さらに、 update 系書き込みの転送先リストも静的に決定されている方が実行時の効率が良い（SMS でも update のコピーページは回収されない）。また、頻度が少なく一部しか外部から参照されないページはコピーページを作らずに遠隔メモリ（home_only ページからの）読み出し（またはプリフェッチ）で対応した方が、性能上もメモリ資源節約上も効率が良い。

大きな粒度でノード間でデータ交換すれば良いデータは **invalidate ページ**に割り当て、 invalidate ページは IVY[6] と同様のページ管理機構を利用した無効化プロトコルベースの仮想共有メモリとして実現される。なお、 update, invalidate, home_only の属性は共有メモリ空間のある一時点においては一意でなくてはならない。つまり、同一の共有メモリ領域に対して、あるタスク[2]ではコピーを持つ update 領域で、他のあるタスクからはコピーを持たない home_only 領域というようなメモリ領域を同一論理アドレスで定義することはできない。ただし、で OS によって同一論理アドレス領域の属性を一斉に変更することは禁止しない。この他に **local ページ**（node_local ではなく processor_local の意）というメモリ領域に対する論理的な属性値が存在し、この属性値の領域はたとえプログラム上の論理アドレスが同一であり同一クラスタ内であってもプロセッサごとに別の独立した実アドレス領域に割り当てられる（つまり別タスクとなる）。この local 属性は集中共有メモリマルチプロセッサのノード（クラスタ）を持つシステムにおいて、 SPMD 型のプログラムを効率良く実行（ローカル変数領域の同一論理アドレス上での確保）するために新たに SSS–CORE のメモリシステムに付加された属性である。この属性のメモリ領域を持つタスクは同一時点では多くとも一つの実行中のスレッド（実プロセッサ）しか持てない。

5 おわりに

現在、ワークステーションクラスタ版 SSS–CORE の開発は Sun Microsystems 社の SPARCstation 10 または SPARCstation 20 を EtherNet で接続した環境で動作している。 WS 単体上の機能はすでに充実しておりメモリ保護、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プログラムロード／実行等が行える。本稿で述べたメモリベース通信機能も一番ベースとなる遠隔メモリ書き込みに関してはすでに実装されており、これを利用した並列レイトレーシングや並列マンデルブロー集合表示のデモプログラムが SSS–CORE 上で動作する。これからメモリベース通信機能を拡充していくと共に、ワークステーション間に跨る資源管理方式やユーザレベルのカーネル協調ランタイム／ライブラリも早急に開発実装していく予定である。また、実装と並行して各種性能データを採取する予定である。

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。ワークステーション版 SSS–CORE の共同開発者である有限会社アックスの駒嵐丈人氏と竹岡尚三氏に感謝いたします。また、本研究に関して有益な議論をいただいた有限会社アックスの渦原茂氏に深謝します。

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, 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).

[3] 松本 尚, 平木 敬: キャッシュインジェクションとメモリベース同期機構の高速化. 計算機アーキテクチャ研究会報告 No.101–15, 情報処理学会, pp.113–120 (August 1993).

[4] 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 (Jay 1992).

[5] 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).