分散共有メモリに基づく計算機クラスタ

平木　敬・丹羽　純平・松本　尚［東京大学］

共有メモリモデルに基づく並列計算

計算機における計算実行は，命令や関数，サブルーチン，タスク，スレッド，オブジェクトなどが互いにデータを授受し，改変することにより実現される．我々が日常使っているフォンノイマン計算機にとり，共有メモリは最も基本的なデータ授受の方法である☆．たとえば，実行される命令間でレジスタとメモリの2種類のメモリ空間を共有することにより，命令実行に必要なデータが授受される．また，サブルーチンや関数間でも，レジスタとメモリを共有することにより引数などが授受される．

並列計算における共有メモリモデルは，逐次計算における共有メモリモデルの自然な延長である．各要素プロセッサで実行される並列プログラムは共有するメモリ（および共有するレジスタ☆2）を介することにより，データの授受を行う（図-1）．

一方，メッセージパッシングモデルでは，通信路を主体として，直接的に通信路に対してsend/receive操作を行うことにより各要素プロセッサ間のデータの授受を実現する．ここでは，接続された通信路を多くの命令，サブルーチン，関数などが共用する．

SMP（集中共有メモリ計算機，図-2（a））は，通信を介することなく直接的に細粒度（ワード単位）の共有メモリを実現するものであり，規模に限界があることとコストが高くなることを除けば最も効率的かつプログラミングが容易な並列計算機であることが広く認められている．一方，SMP以外の計算機，たとえば計算機クラスタ☆3では，各要素プロセッサ間での命令・プログラム間のデータ授受はすべて通信路を介して実現される．したがって，計算機クラスタにおける並列計算では，各要素プロセッサ内における共有メモリを介したデータ授受と，各要素プロセッサ間の通信路を介したデータ授受が混在する．

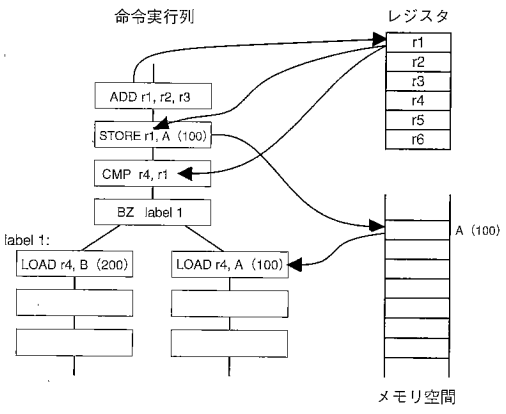

図-1　逐次計算における共有メモリ概念

要素プロセッサがCPU，キャッシュメモリおよびメインメモリを各々持ち☆4，ネットワークにより互いに結合しているシステムにおいて，要素プロセッサ間のデータ授受が共有メモリモデルに基づく方式を分散共有メモリ方式（図-2（b））と呼ぶ．そこでは，共有メモリアクセスが遠隔書き込みまたは遠隔読み出しメッセージに変換されて通信される．一方，データ授受がメッセージパッシングモデルに基づく方式を分散メモリ方式と呼ぶ．データの授受は通信路（メッセージチャネル）に対してパケットのsend/receive操作を行うことで実現される．

しかしながら，計算機クラスタの本質上，各要素プロセッサはマルチプロセス／マルチジョブ状況でありかつ要素プロセッサ間ネットワークは計算機クラスタ内部に閉じていない☆5．したがって，通信の仮想化と保護が必要である．

通信の仮想化は，メモリにおける仮想化と同様に，通信対象の位置を物理的な要素プロセッサ内の特定の通信路やバッファメモリでなく，ユーザプログラムから見た論理的な対象で指定することを意味する．また通信の保護は，ネットワークを介した通信に対して，

☆　命令間でトークンやメッセージ通信を直接的に行う計算モデル，たとえばデータフローモデルやリダクションモデルでは共有メモリの存在を必要としない．

☆2　一部の投機実行を行うアーキテクチャでは，レジスタレベルでの直接的な共有が実現している．

☆3　本稿ではPCクラスタを含めワークステーションクラスタを計算機クラスタと呼ぶ．

☆4　現在のシステムでは，キャッシュメモリの存在を仮定する方が自然であろう．

☆5　イントラネットやインターネットを通じて外部のシステムに接続されている．

要素プロセッサ内部でのデータ授受と同様なアクセス制限を課することである．

分散共有メモリ方式では，仮想化，保護ともメモリ管理ユニット（MMU）などメモリアドレス変換メカニズムとメモリ保護機構により実現される．一方，分散メモリ方式では，仮想化は通信チャネルをメモリ空間へ写像することにより実現され，保護は通信ハンドラをカーネルレベルで実現することにより，プロセッサの特権モードを利用して実現される．

本章でのべた共有メモリモデルは共有データに対するアクセスモデルでなく，データの共有のためのモデルであることに注意する必要がある．すなわち，共有メモリモデルは共有する対象を軸とし，メッセージパッシングモデルは通信路を軸とする．また，具体的なデータの授受は使用するプログラミングモデルにより規定される．また，要素プロセッサ間におけるデータ授受性能は具体的な実現メカニズムに強く依存する．

本稿では，計算機クラスタにおける分散共有メモリ方式の意義，方式，現時点における到達点を示す．

キャッシングと分散共有メモリ

計算機クラスタにおいて，共有データのコピーを，アクセスする要素プロセッサのメインメモリやキャッシュメモリに置くことは非常に有効な高速化の手段である．逐次計算機におけるキャッシュメモリと同様，プロセッサ間で共有するデータに関して時間的（temporal）または空間的（spacial）局所性を利用して共有データに対するアクセス時間を軽減する．しかしながら，書き込みと読み出しが必ず同一プロセッサで実行される逐次計算機の場合と比較して，計算機クラスタでの分散共有メモリアクセスは下記問題点を持つ．

- データを書き込む要素プロセッサと読み出す要素プロセッサが異なる場合，書き込まれたデータを読み出す要素プロセッサに伝達する必要があり，その解決のためのメモリコンシステンシ問題が発生すること．
- 逐次計算機のメモリバスと比較して，著しく低いバンド幅，高いレイテンシのネットワークを使うこと．
- ネットワークの送受信がオペレーティングシステムなどシステムソフトウェアでサポートされているため，メモリアクセスに伴うネットワークの送信，受信が大きなオーバヘッドを伴うこと．

メモリコンシステンシ問題はキャッシュメモリを用いた並列計算に関する問題であるが，計算機クラスタでは，要素プロセッサを結ぶネットワークが十分なバンド幅を持たないケースが多く，メモリコンシステンシモデルの選択が直接的に性能に影響を及ぼす．

コンシステンシ維持の本質は，異なる要素プロセッサで実行しているプログラムからのメモリ同一番地（またはブロック）への読み出しと書き込みで，プログラムが意図した読み出し値が得られることである．異なる要素プロセッサのメモリ上でこのことを実現するためには，

- 書き込みが行われると，次の読み出しまでの間に読み出し側の要素プロセッサの持つメモリブロックを取り消す（インバリデート）か，新たに書き込まれた値に更新する（アップデート）．または，
- 読み出しが行われると，対象の最新の値を読み出し側の要素プロセッサのメモリに得る，

ことが求められる．もちろん，要素プロセッサのメインメモリと同時にキャッシュメモリに対して上記操作が行われれば，さらに高速化することが可能である．

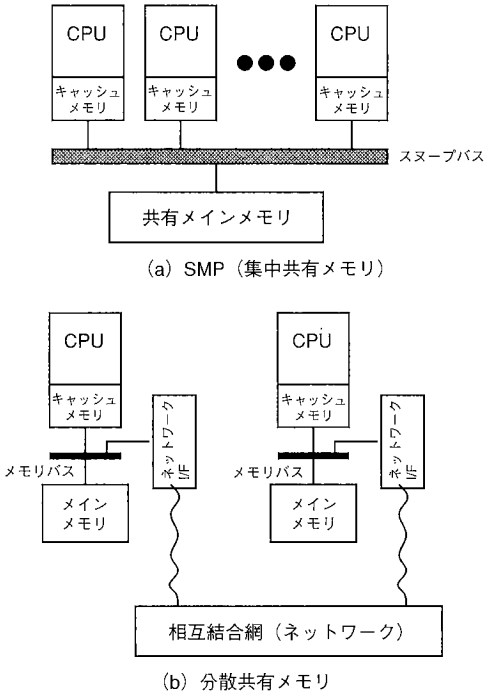

図-2　メモリ共有の基本アーキテクチャ

メモリコンシステンシモデルは，共有メモリアクセスにおいて，アクセス間の因果関係と時間順序の関係を規定するものである[4]．SMPなど要素プロセッサ間の結合が高バンド幅，低レイテンシで実現する場合には，逐次計算機に非常に近いコンシステンシモデル（たとえばシーケンシャル・コンシステンシモデル（SC）[7]）が便利である．そこでは，各要素プロセッサがキャッシュなしで1個の共有メモリを取り合ってアクセスする場合と同じ結果を保証する．しかしながら，計算機クラスタではネットワークのレイテンシが命令実行速度と比較して著しく大きく（1000命令の実行時間以上であることが普通である），通信バンド幅が小さい．実際，最近の高性能CPUを使用したシステムでは，通信レイテンシがCPUにおける1000命令実行時間以上であり，通信バンド幅がメモリバスバンド幅の1/20以下であることが普通である．このような環境では前記コンシステンシモデルに基づく分散共有メモリ方式には下記3種の問題点が発生する．

(A) CPUで実行するメモリアクセス命令，特にストア命令（緩和されたメモリアクセスモデルを用いる場合には，メモリバリア命令）が，ネットワークを介したコンシステンシ維持操作の終了まで完了せず，大きなオーバヘッドを生む．この問題への解決策は以下の通りである：

(A') 緩和されたメモリコンシステンシモデルを使用することにより，コンシステンシ維持操作によるプログラム実行の停止を削減する．

緩和されたコンシステンシモデルでは，因果関係を保証する必要のあるメモリアクセス命令対間に，メモリバリア命令やストアバリア命令などを挿入し，それ以前に発行されたメモリアクセス命令に関するコンシステンシ維持操作の完了を保証する．たとえばWeak Consistency Model（WC）[1]やUltra Sparcプロセッサで採用されているRelaxed Memory Ordering Model（RMO）は，個々のメモリアクセス命令におけるコンシステンシ維持操作完了待ち合わせを要素プロセッサ単位のバリア同期に置き換えることによりオーバヘッドを削減する．

Release Consistency Model（RC）[2]では，コンシステンシ維持に必要な同期をAcquire操作とRelease操作の対に拡張し，操作対象をプロセッサ単位ではなく，メモリ上のコンシステンシ維持操作対象単位として扱う．したがって，実装方式がSCやWC，RMOと比較して遥かに複雑化するが，コンシステンシ維持に伴う同期を最小化し，コンシステンシ維持操作に必要なデータ通信量を削減する．

(B) 実際にはアクセスされないデータに対して同一メモリブロックに存在するために必要のないコンシステンシ維持が発生するfalse sharingや時間的再利用の無視による無駄なコンシステンシ維持のための通信が発生する．この問題点は，下記方式で改善される．

(B') コンシステンシ維持に最小必要限な情報だけを通信することにより，通信量の削減を図る．

Lazy Release Consistency（LRC）モデルは，ネットワークを介した通信量を極小化し，低速な通信の影響を最小限に抑えることを目的として提案され[3),5)]，TreadMarksとして実装された[6)]．

LRCでは，コンシステンシ維持操作をLazyにする，すなわちコンシステンシ維持対象に書き込みが行われた時点ではなく，維持対象が実際に使用される時点にできるだけ近くコンシステンシ維持操作を遅らせる☆6．このことにより，時間的局所性に関するfalse sharingを抑止する．

一方，TreadMarksにおける実現では空間的な意味でのfalse sharingは，メモリの共有アクセス前の状態を保存するコピーであるtwinを作成し，コンシステンシ維持操作時にtwinとの差分を検出する操作（diff）により，メモリ内容の差分だけを通信し通信データ量を抑制する．

(C) コンシステンシ維持操作が，プロセッサの命令がアクセスするデータ単位であるため，細粒度通信が数多く発生し，通信に伴うオーバヘッドの影響を強く受け性能が低下する．この問題点は下記方式で改善可能である．

(C') LRCモデルを用いることにより，コンシステンシ維持のための通信のブロック化が可能となる．

また，後述する非対称分散共有メモリ（Asymmetric Distributed Shared Memory, ADSM）では，最適化コンパイラによる通信のブロック化，たとえばアクセスにおける連続アドレスの検出による通信の粗粒度化やソースプログラムからみて必要なコンシステンシ維持だけに限定した通信により，さらに通信量，特に通信回数と同期回数を削減する[11)]．

計算機クラスタを構成するテクノロジ

本章では，今日の高性能高並列計算機を実現する要素技術の概要を示す．

自明なことではあるが，高性能計算機の高速化は要素プロセッサの高速化とプログラム実行時におけるプロセッサ並列度の積として求められる．

要素プロセッサの高速化は，基本クロックの向上と，プロセッサ内部の命令レベル並列性向上で実現される．その双方にとり，命令間データ授受の手段であるプロセッサ内共有メモリの実現方式が性能に対し非常に大きなインパクトを持つ．

要素プロセッサ内部での共有メモリ実現に必要な機能，すなわち，仮想メモリの実現，ページ単位のメモリ保護の実現はTLB（Translation Lookaside Buffer）を含むメモリ管理機構（MMU）で実現される．また，SMPを構成するために必要な並列キャッシュ機構は，キャッシングされたデータに対するアクセスをキャッシュ外部へ反映させるために必要な機構である．MMUと並列キャッシュ機構により，CPUで実行されるプログラムと外部とのデータの授受が，保護と仮想化を実現しつつ命令実行レベルの速度で実現される．計算機クラスタの高速実現にとって，これらメモリ管理機構の利用は採用する通信モデルにかかわらず不可欠である．

並列／分散計算では，ネットワークは要素プロセッサ間通信を実現するために必要である．ネットワーク性能，すなわち通信レイテンシと通信バンド幅の向上は，ただちに並列分散計算の性能向上を意味する．ここで問題になる通信性能は，通信媒体としてのピーク性能ではなく，実際に異なる要素プロセッサでのユーザ間で授受される場合の性能である．多くの並列専用

☆6 LRCはRelease Consistencyモデルに基づいているため，実際のメモリアクセスではなく，AcquireとRelease操作に伴うコンシステンシ維持操作をLazyにする．

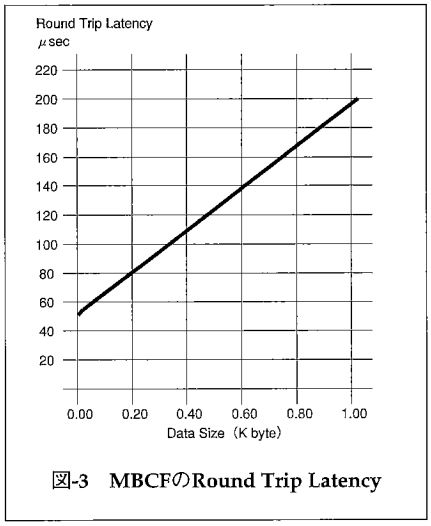

図-3 MBCFのRound Trip Latency

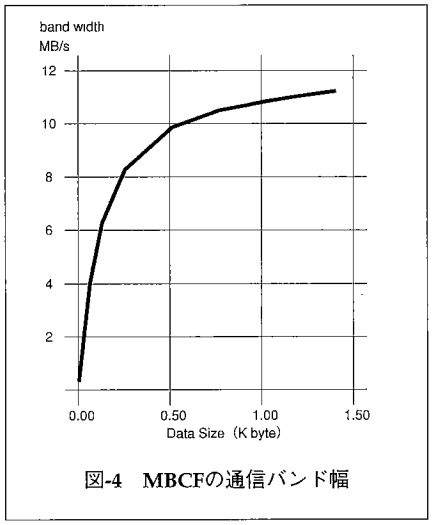

図-4 MBCFの通信バンド幅

計算機では，内部ネットワークはすべてハードウェア制御で実現されるため，ネットワーク媒体本来の性能を引き出すことが可能である．

近年の急速なネットワーク媒体性能の向上は著しく，要素プロセッサの性能の向上を大きく凌駕している．たとえばギガビットイーサネットやMyrinetでは，ネットワーク媒体自身の性能として1ギガビット／秒以上の通信バンド幅，数100ナノ秒の通信レイテンシを実現する．

しかしながら，計算機クラスタでは通信の起動，送信，受信にソフトウェア，特にカーネルレベルのソフトウェアが介在する．その結果，ネットワーク媒体が持つ基本性能の極一部しか，実際のユーザレベル間の通信では得られない．たとえば典型的なUNIX上でのTCP/IP通信では，ネットワーク媒体に100BASE-TXイーサネットを用いると，約700マイクロ秒の通信レイテンシ，約6メガバイト／秒の通信バンド幅しかユーザ間では得られない（規格からの最大性能は12.28メガバイト／秒，6マイクロ秒である）．

これら性能低下の原因は，入出力バスの起動手順，データバッファのコピーやシステムコールを実現するために実行される余分な命令数（命令オーバヘッド），受信のための割り込み処理にかかわるオーバヘッドなど多種にわたる．いずれもオーバヘッドは実行速度にほぼ反比例する性質のものである．したがって，ネットワーク媒体の性能向上が要素プロセッサの性能向上を上回る状態では，通信のためのソフトウェアによる命令オーバヘッドが性能を支配し，ネットワーク媒体の性能向上はそのままではユーザが享受できない．また，この命令オーバヘッドのため，通信パケット長が短い場合著しく性能が低下する．UNIXにおける100BASE-TXの場合，最大バンド幅の半分の転送能力が得られるパケット長は256バイト前後であり，そのままでは細粒度通信となる分散共有メモリとの相性が非常に悪い．

このことは，同時に計算機クラスタの高性能化に対するオペレーティングシステムの重要性を示している．現在実用に供されているオペレーティングシステムは，入出力の1つとしてネットワークをサポートしているため，要素プロセッサ間のデータ授受に対するオーバヘッドが大きく，またユーザに対し単一計算システムとしてのイメージを与えることに困難がある．

計算機クラスタ上に共有メモリを実現する方式

プロセッサが持つ仮想メモリ管理機構を利用したネットワーク方式，最適化コンパイル技術，およびオペレーティングシステム技術の急速な発展により，共有メモリを計算機クラスタで実用のプログラミングモデルとして用いることが可能となりつつある．

仮想メモリ管理機構（MMU）のネットワーク通信への利用は，汎用ハードウェアを用いたソフトウェア分散共有メモリを可能とする[8]とともに，通信のためのバッファへのデータコピーと通信を起動するための命令オーバヘッドを削減した．

並列／分散プログラムから見ると，通信はプログラムがアクセスするメモリ間のコピーに他ならない．通信対象メモリブロック（多くの場合メモリページ）のアクセスにより通信操作を起動する．管理の単位がページレベルの場合，MMU内にあるTLB上のページ保護情報などを利用して，割り込みを発生し，ネットワークを介したコピー動作を起動する．受信側では，ネットワークからの割り込みにより通信されたデータをメモリにコピーし，ページ保護属性などの状態を適切に設定する．したがって，保護と信頼性の実現を無視すれば，通信に附属するハードウェアバッファへのコピー以外の無駄がない0コピー通信が実現可能である．ただし，0コピー通信において信頼性（到着性，順序保存性）を保証し，かつ通信と実行の並行動作を実現するためには，ライトアフターライトハザードを避けるための同期を挿入する必要があり，かえって通信と実行の並行動作を妨げる．さらに，保護の実現の観点からも特殊なネットワークインタフェースを用い，通信チャネルを固定した通信を実現しない限りネットワーク通信に関する保護が成立しない．1コピー通信は，1回だけ送信時にカーネルが持つ通信バッファにコピーを行うことにより，保護され，信頼性を持った通信を，通信と実行の並行動作を保ったまま実現する．

Memory-Based Communication Facility (MBCF)[9]は，1コピー通信による仮想化，保護，信頼性を実現した通信方式であり，**図-3**と**図-4**に示すように，100BASE-TXイーサネットの性能を純ソフトウェアで引き出している．MBCFでは通信の起動，通信先へのアドレス変換，保護の実現がCPUの持つMMU機能により実現されるため，オーバヘッドが少ないことが示されている．

メモリコンシステンシの維持にソフトウェアで実現する通信を利用すると，前記のようにプログラムでの

ロード，ストア命令単位では通信のデータサイズが細かすぎ，プログラムの進行がコンシステンシ維持操作により長期間停止するため低い通信性能しか得られない．しかしながら，コンシステンシ維持の単位を大きくとるとfalse sharing，すなわちプログラムが実際に使用しないデータにまでコンシステンシ維持が行われ，そのための通信量が増大し性能が低下することが発生する．

LRCに基づくTreadMarksまたはその延長線上にあるAURC[3]はtwin/diff操作や通信操作に伴う非常に大きい命令オーバヘッドにもかかわらず，スケーラブルな性能を示し，計算機クラスタにおけるソフトウェア分散共有メモリ方式の実用化を示唆した．

TreadMarksなどロード，ストア命令を直接使用するソフトウェア分散共有メモリ方式では，通信データ量の極小化は，ロードとストアにおけるアドレス列という下位レベル情報を基に行われる．プログラムと分散共有メモリとのインタフェースをコンパイラが生成するコンシステンシ維持操作の直接起動を含む命令列まで拡大すると，割り込みにかかわるオーバヘッドを削減できるとともに，コンパイラによる最適化をさらに押し進めることが可能となる．我々が研究開発を進めている非対称分散共有メモリ（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分散共有メモリである[11]．ADSMは，共有メモリからの読み出しは，プログラム中のload命令とMMUによる割り込みにより共有動作を起動し，一方書き込みは直接コンパイラがコンシステンシ維持コード，すなわちメモリ状態の管理，同期，通信などのコードをコンパイラが直接プログラムに埋め込むことを特徴としている．ADSMで実現している最適化は下記に示す通りである．

- ページ単位のプロトコル切替え

コンパイラがストアの後に挿入するコンシステンシ維持コードを切り替えることによって，さまざまなプロトコルを実現

- コンシステンシ維持コード列のコアレシング（coalesing）

ある同期区間において連続した共有アドレスへの書き込み列がある場合，コンシステンシ維持コードの列を，連続領域に対する1つの維持コードに変換

- 通信パケットのコンバイニング

送り先が同じパケットをコンバイニングしてメッセージ数を減らし，更新型メモリアクセスのオーバヘッドを削減であり，通信を粗粒度化するとともに，不要な同期操作を除去し，複数プロトコルの自由な混在を実現する．

一方，Eager RCをソフトウェアで実装したShasta[12]では，アセンブラレベルのコード解析によって局所領域以外のアクセスにコンシステンシ維持コードが挿入される．

性能評価

本章では，計算機クラスタにおける分散共有メモリが到達した性能レベルを，並列計算機との比較で示す．

計算機クラスタとして，SUN SparcStation 20を100BASE-TXイーサネットとハブスイッチで結合したシステム，並列計算機として富士通AP-1000+を用いた．被評価プログラムはSPLASH-2[13]の中のKernelのLU-Contigを用い，データのhomeの配置の最適化を行った．評価はプログラムをまずADSMコンパイラで処理し，引き続きバックエンドコンパイラとしてgccを用いた．

- **AP-1000+**

AP-1000+は，要素プロセッサを専用内部ネットワークで結合した並列計算機である．各要素プロセッサは50MHz SuperSPARC（二次キャッシュなし，16MBytesメモリサイズ）でネットワークは2次元トーラスでバンド幅は各リンクごとに25MBytes/secである．AP-1000+のOSはユーザレベルのページフォールトハンドラをサポートしていないため，今回の実装では，共有ページへのアクセスの前にページの有効性をチェックするコードを挿入した．有効でないページにアクセスしたら，ユーザレベルのページフォールトハンドラを呼ぶ．

図-5は静的なコアレシングの効果を調べたものである．**S**はコアレシングしたもの，**N**はしないものを示す．問題のサイズは256×256行列でブロックサイズが16である．グラフの縦軸は実行時間（秒），横軸はプロセッサ台数である．図中，taskはアプリケーションの実質的な実行時間，msgはメッセージ処理時間，PFはページフォールト処理時間，CMはコンシステンシ維持コードの実行時間，syncはロックまたはバリア同期処理の時間である．静的なコアレシングにより，ボトルネック部分のオーバヘッド軽減に成功していることが示されている．

- **計算機クラスタ**

計算機クラスタのノードはSun SS20（85MHz SuperSPARC×1）でFast Ethernet SBus Adapter 2.0をつけて，100BASE-TXをスイッチで相互接続している．ノード間は**MBCF**で結合されている．計算機クラスタでは汎用超並列オペレーティングシステムSSS-COREを使用する．

図-6はAP-1000+との比較を行った結果である．プロセッサ台数は8台まで変化させた．縦軸は実行時間である．

今回採用したプロトコル（Software-implemented AURC, SURC）はネットワークに対する負荷が大きいため，ネットワークが高速なAP-1000+に有利であるが，SSS-COREのメモリベース通信機能が十分性能を発揮しているために，実行時間の比はプロセ

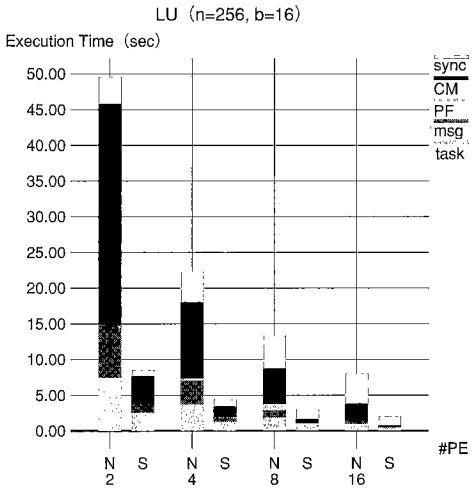

図-5　静的なコアレシングの効果（AP-1000+）

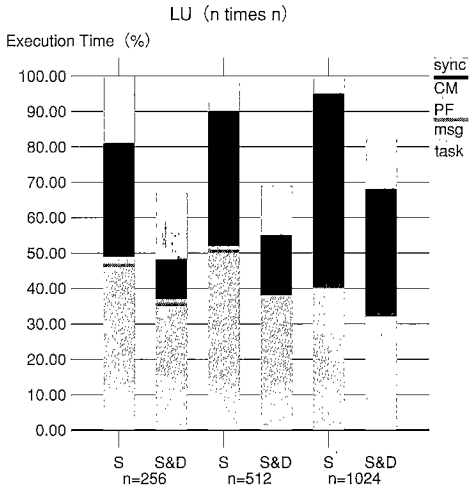

図-7　動的なコアレシングの効果（SSS-CORE）

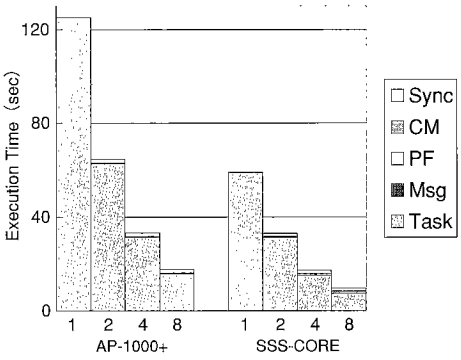

図-6　SSS-COREとAP-1000+の比較

ッサのクロック比に近い値になっている．

図-7は動的なコアレシングの効果を調べたものである．**S**は静的にコアレシングしたもの，**S&D**は静的かつ動的にコアレシングしたものを示す．プロセッサ台数は4台で，行列の一辺のサイズを256，512，1024と変化させた．縦軸は実行時間で静的にコアレシングしたもので正規化している．動的にコアレシングを実行することで実行時間の削減に成功している．動的にコアレシングを実行した方が，コンシステンシ維持コードの時間（CM）の削減に成功している．

これらの実験により，ADSMコンパイラが用いた最適化の有効性が示されているとともに，計算機クラスタを用いた並列計算が，ほぼ専用ネットワークを持った並列計算機のレベルに達していることが示されている．

今後の方向性とまとめ

コンパイラの最適化技法，計算機クラスタを目的とした高性能オペレーティングシステム，メモリベース通信機構と，プロセッサが持つ仮想メモリ管理機構の活用による，汎用ネットワーク（イーサネット）で結合した計算機クラスタにおける分散共有メモリ方式はすでに実用的レベルに達している．

本稿では触れなかったが，汎用ネットワーク上に低オーバヘッドなメッセージパッシングを実現するためにも，プロセッサが持つ仮想メモリ管理機構を活用する分散共有メモリのフレームワークを用いる方が性能的に有利である[10]．また，大規模計算機クラスタでは，使用状況の変化によるプロセスのマイグレーションを含む動的な使用形態が望まれ，計算機クラスタの基本アーキテクチャとしての分散共有メモリの果たす役割は大きい．ギガビットイーサネットから，10ギガビットイーサネットが普及する近未来には，コンパイラが直接サポートする分散共有メモリが計算機クラスタの基本アーキテクチャとなることを期待する．

参考文献

1) Dubois, M., Scheurich, C. and Briggs, F.: Memory Access Buffering in Multiprocessors, Proc. 13th Int. Symp. on Computer Architecture, pp.434-442 (1986).
2) Gharachorloo, K., Lenoski, D., Laudon, L., Gibbons, P. and Gupta, A.: Memory Consistency and Event Ordering in Scalable Shared-Memory Multiprocessors, Proc. 17th Int. Symp. on Computer Architecture, pp.15-26 (1990).
3) Iftode, L., Dubnicki, C., Felten, E.~W. and Li, K.: Improving Release-Consistent Shared Virtual Memory using Automatic Update, Proc. of the 2nd Inter. Symp. on HPCA (1996).
4) 城　一貴: メモリコンシステンシモデル（チュートリアル），JSPP '97予稿集, pp.85-90 (1997).
5) Keleher, P., Cox, A.~L. and Zwaenepoel, W.: Lazy Release Consistency for Software Distributed Shared Memory, Proc. of the 19th ISCA, pp.13-21 (1992).
6) Keleher, P., Dwarkadas, S., Cox, A.~L. and Zwaenepoel, W.: TreadMarks: Distributed Shared Memory on Standard Workstations and Operating Systems, Proc. of the Winter 1994 USENIX Conference, pp.115-131 (1994).
7) Lamport, L.: How to make a Multiprocessor Computer that Correctly Executes Multiprocess Programs, IEEE Trans. on Computers, Vol.~C-28, No.~9, pp.690-691 (1979).
8) Li, K.: IVY: A Shared Virtual Memory System for Parallel Computing, Proc. of the 1988 ICPP, pp.94-101 (1988).
9) Matsumoto, T. and Hiraki, K.: MBCF: A Protected and Virtualized High-Speed User-Level Memory-Based Communication Facility, Proc. of Int. Conf. Supercomputing (1998).
10) Morimoto, K., Matsumoto, T. and Hiraki, K.: Implementing MPI with the Memory-Based Communication Facilities on the SSS-CORE Operating System, Proc. 5th European PVM/MPI User's Group Meeting (LNCS Recent Advances in Parallel Virtual Machine and Message Passing Interface), pp.223-230 (1998).
11) Niwa, J., Inagaki, T., Matsumoto, T. and Hiraki, K.: Efficient Implementation of Software Release Consistency on Asymmetric Distributed Shared Memory, Proc. of Int. Symp. on Parallel Architectures, Algorithms and Networks (I-SPAN 97), pp.198-201 (1997).
12) Scales, D.~J., Gharachorloo, K. and Thekkath, C.~A.: Shasta: A Low Overhead, Software-Only Approach for Supporting Fine-Grain Shared Memory, Proc. of 7th Int. Conf. on ASPLOS, pp.174-185 (1996).
13) Woo, S.~C., Ohara, M., Torrie, E., Singh, J.~P. and upta, A.~G.: The SPLASH-2 Programs: Characterization and Methodological Considerations, Proc. of the 22nd ISCA, pp.24-36 (1995).

(平成10年9月28日受付)