ネットワークへのオフロードに基づいた OpenMP の高速実行方式

會田貴裕[†] 栗林肇[†]
安田浩司[†] 和田耕一[†]

近年，潜在的に高い計算能力を持つクラスタ上で，OpenMP などに代表される共有メモリパラダイムに基づいたプログラムを実行可能とする分散共有メモリ (DSM：Distributed Shared Memory) の研究が精力的に進められている．DSM を実現するには，DSM 領域に書き込まれたデータを，各プロセッサが正確に読み出せなければならない．そのための手法として，参照する共有データの依存性を実行時に解析し，プロセッサ間で依存データの授受を行うことで，データの一貫性を保証する方法がある．本プロジェクトは，この共有データへのアクセス依存性解析処理や通信に関する処理をクラスタ向けネットワークへオフロードすることで，OpenMP によって記述されたプログラムを高速に実行できるシステムを実現するものである．

The high-speed-execution system of OpenMP based on offloading to the network

Takahiro Aida,[†] Hajime Kuribayashi,[†] Koji Yasuda[†] and Koichi Wada[†]

In recent years, Distributed Shared Memory (DSM) which enables execution of the program based on the shared memory paradigm represented by OpenMP etc. on the cluster which has high performance potentially is studied energetically. In order to implement DSM, each processor must be able to read correctly the data written to the DSM area. For that purpose, it is necessary to maintain the consistency of data. One of the methods which maintains it is analyzing the dependability of the shared data referred at the time of execution and transferring dependent data between each processor. This project implements the system which can execute the program described by OpenMP at high speed by offloading analysis processing of access dependability and communication to the network for clusters.

1. はじめに

クラスタ上で並列プログラミングを行う際，その分散配置された構成の特性上，MPI[6] をはじめとするメッセージパッシングパラダイムが利用されることが多い．しかし，メッセージパッシングパラダイムにおいてはプログラマがデータの所在を把握・制御する必要があり，複雑なコーディングになる場合がある．一方，OpenMP などに代表される共有メモリパラダイムでは，プログラマは通信を明示的に記述する必要がないため，プログラミングが容易であると言える．近年計算機クラスタ上で共有メモリパラダイムに基づいたプログラムを実行可能とするシステムの研究が精力的に進められている．このような研究は，分散共有メモリ (DSM：Distributed Shared Memory) として知られている．

DSM を実現するには，DSM 領域に書き込まれたデータを，各プロセッサが正確に読み出せなければならない．そのための手法として，参照するデータの依存関係を実行時に解析し，プロセッサ間で依存データの授受を行うことで，データの一貫性を保証する方式がある．しかし，依存性解析のための演算処理などが全体の実行時間に対してオーバヘッドとなっていることが分かっている[7)4)]．また，依存データ授受のための通信が，アプリケーションによっては 1 対全通信や全対全通信になる場合があり，これらの集団通信がオーバヘッドとして顕在化することもある．

† 筑波大学大学院システム情報工学研究科コンピュータサイエンス専攻
Department of Computer Science, Graduate School of System and Information Engineering, University of Tsukuba

筑波大学並列分散処理研究室（以下本研究室と呼ぶ）で開発した Maestro cluster network は，通信モジュールを実装する FPGA と汎用プロセッサとを密に結合したアーキテクチャにより高い通信性能を実現する．また，Maestro は従来各ホストプロセッサで行っていた演算処理や通信処理をネットワークインタフェース (NI) やスイッチへオフロードする機構を持つ．これにより，ホストプロセッサと NI との並列処理や，スイッチの機能を用いた高度な集団通信を可能とする[8]．

本プロジェクトでは，従来の実装においてホストプロセッサで行っていた DSM 実現のための依存性解析処理を，Maestro の NI へオフロードし，さらに，Maestro のスイッチの機能を用いて実装した集団通信を適宜使用することによって，OpenMP によって記述された共有メモリ型プログラムの高速な実行方式を提案する．

2. DSM 実行時ライブラリによる共有メモリ環境

本節では，計算機クラスタに代表される分散メモリ環境において共有メモリ空間を実現する方法について述べる．さらに，本研究室提案の配列範囲記述子 quad を用いた DSM 実行時ライブラリによる共有メモリ環境について説明する．

2.1 分散メモリシステム上での共有メモリパラダイム

一般的に計算機クラスタは分散メモリ環境であることから，プロセッサは自身のローカルメモリにはアクセスできるが，他のプロセッサのメモリには直接アクセスすることはできないため，ネットワークを介したアクセスとなる．つまり，物理的にメモリ空間を共有することができない（図 1）．

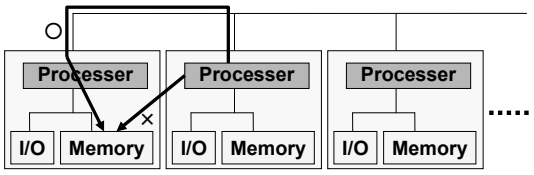

図 1 分散システムのメモリアクセス

そこで，このような分散システム上に仮想的な共有メモリ空間を定義することによって共有メモリパラダイムを利用可能にする．このような仮想的な共有メモリシステムを分散共有メモリ (DSM) と呼ぶ（図 2）．

DSM を計算機クラスタ上で実装するひとつの手法

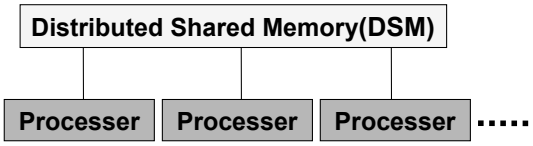

図 2 分散共有メモリ

として，アクセス対象のデータを持っているプロセッサが，そのデータを読み出すプロセッサへ，読み出し前にあらかじめデータを送信しておくことが挙げられる．また，共有メモリパラダイムであるということは，この時に行われる転送データの特定及び通信はプログラマには透過的に行われる必要がある．この透過性を実現するための一つの手法として，コンパイラによってデータアクセス情報管理・解析のコードを追加する実装が挙げられる．

2.2 配列範囲記述子 quad

一般的なアプリケーションでは配列に格納された大量のデータを扱うため，配列の各々の要素について依存関係を調べて転送の必要性の有無を特定していると処理が膨大になってしまう．そこで，本研究室で提案の quad を利用することによってその問題を解決する[3]．quad は配列のアクセス範囲を簡潔に 4 つの数字の組み合わせによって表現する．i) 先頭領域の配列オフセット，ii) アクセス領域の長さ，iii) 非アクセス領域の長さ，iv) 繰り返し数というパラメータの組み合わせである．メモリ領域の先頭から 3 つの領域にアクセスを行い，1 つの領域をアクセスしないという操作を 3 回繰り返す場合の quad は (0,3,1,3) である（図 3）．

図 3 quad の例

プログラム実行時に依存データを特定するには，読み出されるデータと書き込まれたデータを表現する quad 間で積演算 (intersection) を行えばよい．また quad 間の和演算 (union) は，いくつかの quad をより少ない数の quad にまとめるために用いられる（図 4）．

2.3 quad を用いた DSM 実行時ライブラリによる共有メモリ

計算機クラスタと quad を用いて共有メモリ型プログラムを実行する手順について具体的な手順を図 5 を用いて説明する．まず，共有メモリ型プログラムに記

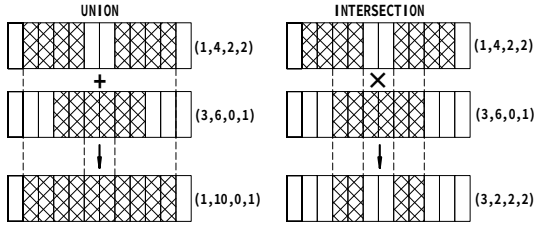

図 4 quad 演算の例

述されているメモリアクセスを quad で管理するため，プログラムをコンパイルすることで，プログラムのアクセスパターンを解析して，1) アクセスされるデータ，2) アクセスされる範囲，3) アクセスするプロセッサ，4) アクセスタイプ，などの情報を得る．この情報をもとに，DSM 実行時ライブラリが用意している関数を追加する．コンパイラによって関数が追加された共有メモリ型プログラムを計算機クラスタ上で実行する際，以下の作業が行われる．

(1) **依存性解析**

以下の （a）と（b）の作業を行うことで，実行アプリケーションで扱う DSM 領域のデータについての依存性解析を行う．

(a) **quad 登録**

コンパイラによって quad 登録コードが挿入された共有メモリプログラムを実行すると，データの読み書きをしたことを表す read quad 関数及び write quad 関数が実行される．各プロセッサは，これらの quad をそれぞれの持つメモリ上に作成した quad テーブル上にプロセッサ ID とアクセスパターンの種類ごとに分別して登録・管理する．

(b) **quad 演算**

各プロセッサは push 関数と呼ばれるメモリ領域の依存性解析・データ転送を行う関数を呼ぶ．push 関数では，（a）で登録した quad の登録情報に基づいて，依存性解析を行い，送受信すべきデータ領域を特定する．

(2) **依存データ送受信とプロセッサ間同期**

引き続き push 関数は，(1) の解析により求められた依存関係にあるデータを受信側のプロセッサに転送する．そして，各プロセッサで受信が行われた後に同期を取ることによって，全プロセッサが最新の共有メモリ領域のデータを持っていることが保証される．

本研究室では，共有メモリ型プログラムである OpenMP プログラムに quad の生成，登録，演算のためのコードを追加するコンパイラとして，Quaver[5] を開発している．Quaver でコンパイルされたプログラムは，DSM 実行時ライブラリが用意している関数が

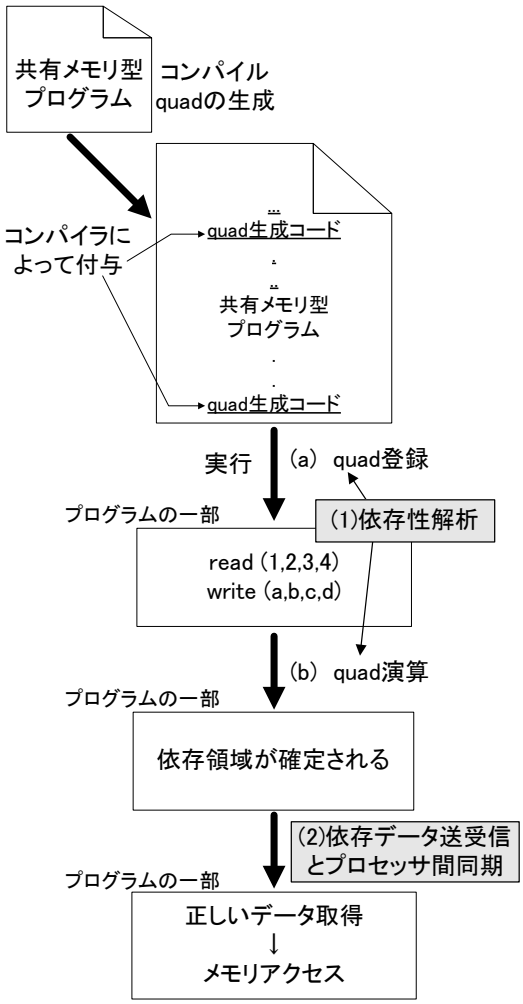

図 5 共有メモリ型プログラム実行手順

追加され，更に MPI 用コンパイラによってコンパイルされることによって実行可能ファイルに変換される．

3. Maestro cluster network

本研究室では，Maestro cluster network と呼ぶクラスタ向け高性能ネットワークを開発している．Maestro は，従来クラスタネットワークとして多用されている中広域ネットワーク向け通信ネットワークデバイスの機能，性能と，クラスタ内通信に必要とされる機能，性能との差に注目し，クラスタ内通信で求められる低レイテンシ，高スループットを実現するネットワークの構築を目指して開発されたものである．

Maestro シリーズの第 2 世代機である Maestro2 は NI とスイッチボックス (SB) によって構成される (図 6)．NI はホストプロセッサと PCI によって接続され，SB は各 NI と最大 8 台まで接続される．Maestro cluster

network の主な特徴として以下が挙げられる．

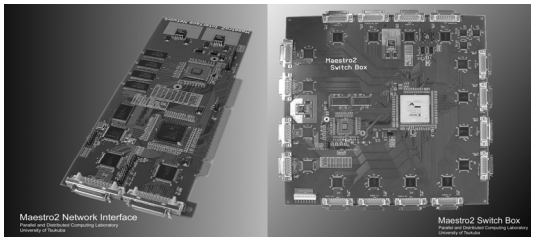

図 6 Maestro2NI(左) と SB(右)

(1) **通信プロトコル処理のオフロード**

Maestro cluster network 向けに開発された通信ライブラリ MMP[9] により，ホストプロセッサからの通信要求の取得をはじめ，送信時のメッセージヘッダの付加，受信時のヘッダの解析，転送対象バッファの仮想物理アドレス変換，DMA コントローラの制御など，通信プロトコル処理の大部分をネットワークデバイスへオフロードすることができる．ホストプロセッサでは，NI への転送要求発行と，転送要求完了通知の受領のみを行う．またクラスタ内のホストプロセッサ間通信として，MMP がサポートするのは 1 対 1 通信のみである．

(2) **ゼロコピー通信**

アプリケーションプログラムからメッセージの送信を行う場合，MMP は，メッセージをホスト PC で実行されるアプリケーションプログラムのデータ領域から，直接ネットワークインタフェースのネットワークバッファに DMA 転送を行う．受信の場合も同様に，アプリケーションプログラムが用意する受信バッファに，ネットワークインタフェースのネットワークバッファからメッセージを直接 DMA 転送する．このゼロコピー通信を行う際に，MMP はアプリケーションプログラムの用意するメッセージの送受信バッファが含まれるメモリページをピンダウンし，オペレーティングシステムのページング対象からはずして，必ず同じ物理メモリアドレスに存在するように固定する．この操作によって，ネットワークインタフェースが，オペレーティングシステムを介することなくアプリケーションプログラムのバッファにアクセスすることができるようになり，ゼロコピー通信を可能とする．

(3) **Maestro Dynamic Offloading 機構**

NI・SB には汎用プロセッサが搭載されている．これを利用し，ネットワークコンポーネントに対してプログラムモジュールを動的オフロードする機構を Maestro Dynamic Offloading(MDO) 機構[1] と呼ぶ．MDO 機構を用いることで，従来ホストプロセッサにて行われてきた演算処理の一部を NI・SB 上で実行可能とし，オフロード対象の演算処理と，ホストプロセッサにおけるオフロード対象外の演算処理とを重ね合わせることができる．MDO 機構では，オフロードするプログラムモジュール (以下モジュールと呼ぶ) 内において，オフロード対象の演算処理の他に，前述した通信ライブラリ MMP の内部で行われていた一連の通信処理を，任意のタイミングで個々に制御することができる．また，1 対全，もしくは全対全の通信が必要な集団通信においては，SB に搭載されているクロスバスイッチを制御するモジュールを登録することで，ある NI から送信されたメッセージを他のすべての NI に対して低オーバヘッドでブロードキャストすることができる．

4. 設計と実装

4.1 従来手法における問題点

先行研究[7][4] によって，OpenMP プログラムをコンパイルして DSM 実行時ライブラリの関数群を追加し，分散メモリ環境上での性能評価を行った．しかし，これらの実装には以下の点においてオーバヘッドや最適化の余地があることが分かった．

(1) **通信プロトコル処理のオーバヘッド**

通信ライブラリとして Gigabit Ethernet 上の MPI を採用しているため，通信プロトコル処理はホストプロセッサにより逐次的に行われている．これにより，依存データ授受に要する通信時間は，実行台数と共に増加する．

(2) **依存性解析処理のオーバヘッド**

quad を用いて実現する DSM では，プログラム実行時に，quad 登録や quad 演算を用いて行う依存性解析が必要となる．しかし，総実行時間中に占める依存性解析処理時間の割合が，実行ホスト数やアプリケーションプログラムによっては無視できないほど大きくなる．

(3) **非効率な集団通信**

1 対 1 通信を基本とした実装であるため，依存データ授受のための通信パターンが 1 対全や全対全である場合，通信オーバヘッドが大きい．

4.2 Maestro を用いた DSM の設計

4.1 で示した問題点を解決するために，第 3 節で説明した Maestro cluster network へのオフロードに基づく実装を行う．

4.2.1 通信プロトコル処理のオフロード

Maestro に実装されている通信ライブラリ MMP を用いて，通信プロトコル処理のオーバヘッドを解決す

る．本提案について図 7 を用いて説明する．「従来手法 MPI」の依存データ通信や，同期処理における通信で発生する通信プロトコル処理は，各ホストプロセッサにおいて逐次的に行われ，かつオーバヘッドも大きい．これを MMP ライブラリを用いて実装することで各通信処理に要する時間を削減し，ホストプロセッサの処理との重ね合わせを行う．すなわち，Maestro cluster network を用いることで，送受信データのコピー操作を最小限に抑えることによって通信処理を削減すると共に，ホストプロセッサのみで逐次的に行われてきた通信プロトコル処理をネットワークインタフェースにオフロードすることで，通信処理時間の隠蔽を図る．

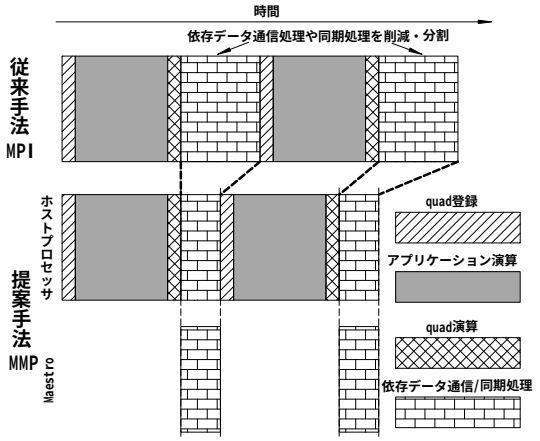

図 7 通信プロトコル処理のオフロード

4.2.2 依存性解析処理のオフロード

Maestro cluster network に実装されている動的オフロード機構 MDO 用いることによって依存性解析処理のオーバヘッドを解決する．本提案について図 8 を用いて説明する．4.2.1 で述べた MMP のみによる実装では，ホストプロセッサが quad 登録，アプリケーション演算，quad 演算を逐次的に行った後に Maestro2 へ通信命令を発行していた．これに対して本節の提案手法では，quad 登録や quad 演算の依存性解析処理をモジュールとして Maestro2 のネットワークインタフェース (NI) へオフロードする．モジュールは，ホストプロセッサで行われるアプリケーション演算と重ね合わせて依存性解析処理を行い，次回の通信においてどのような情報授受が必要かを並行して求める．ホストプロセッサはアプリケーション演算を完了するとモジュールへ完了を通知し，モジュールは既に求めている送受信情報に基づき，依存データの送受信をリモート DMA 通信によって行う．そして通信が完了するとモジュールはホストプロセッサへ完了を通知し，ホストプロセッサとモジュールは次のフェーズへそれぞれ移行する．ここでの送受信の際，MMP による実装と MDO 機構による実装のどちらであっても，ネットワークインタフェース主動の通信であることには変わりないが，MMP での実装の場合は必要回数分通信命令を発行しなければならないことに対して，MDO 機構を用いた実装の場合は，送受信を行うために必要なすべての情報を，送受信の開始時点でモジュールが持っているため，ホストプロセッサとモジュール間の通信のトリガーとなる情報の授受は，前述したように送受信フェーズの最初と最後の通知のみでよい．
本設計により，従来はホストプロセッサにて逐次的に行ってきた依存性解析処理のオーバヘッドを隠蔽し，アプリケーション実行時間の削減に繋がると考えられる．

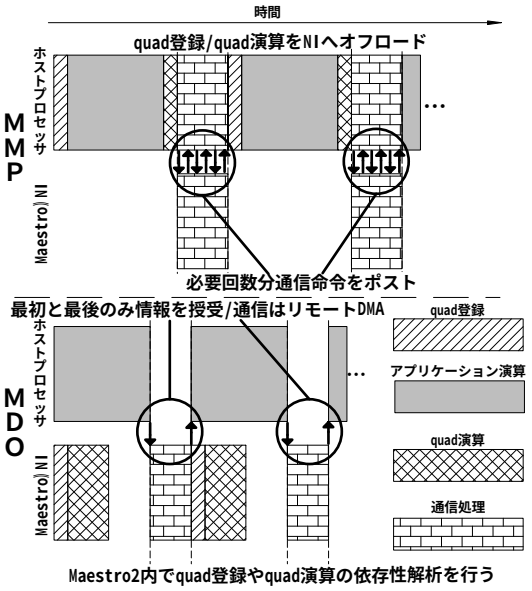

図 8 依存性解析処理のオフロード

4.2.3 集団通信処理のオフロード

非効率な集団通信を解消するための設計を図 9 を用いて説明する．第 3 節で記したように，MDO 機構を用いることで，Maestro のスイッチボックスのクロスバを操作しオーバヘッドの少ないブロードキャストを行うことが可能である．しかし図 9 の上の例のように，従来はアプリケーションが 1 対全の通信パターンを示す場合であっても，1 対 1 通信の繰り返しにて通信部を実装していた．これに対して，MDO 機構を用いてスイッチボックスのクロスバを操作し，送信メッセージのブロードキャストを行うことで，図 9 の下の例のように Maestro により適した通信方法を実装す

る．また，全対全通信についてはホスト数分の回数のブロードキャストを行うことにより実装する．本実装においては，4.2.2 の依存性解析処理に加え，発生する依存データの通信がブロードキャストによって送受信されるべきかを解析するための，通信パターン解析処理もネットワークインタフェースへオフロードする．その処理も依存性解析処理と同様に，ホストプロセッサのアプリケーション演算と重ね合わせて行うように実装する．この解析処理はプログラマにとって透過的に行われるため，様々なアプリケーションプログラムの個々の通信パターンに対して，プログラマ自身が最適な通信を行うための分析を行うことや，特別なコードを追加する必要はない．

本設計により，従来は 1 対 1 通信の繰り返しにて行われていたブロードキャストが，1 ステップで送信を完了できるようになる．このことで通信時間を削減でき，アプリケーション実行時間の削減に繋がると考えられる．

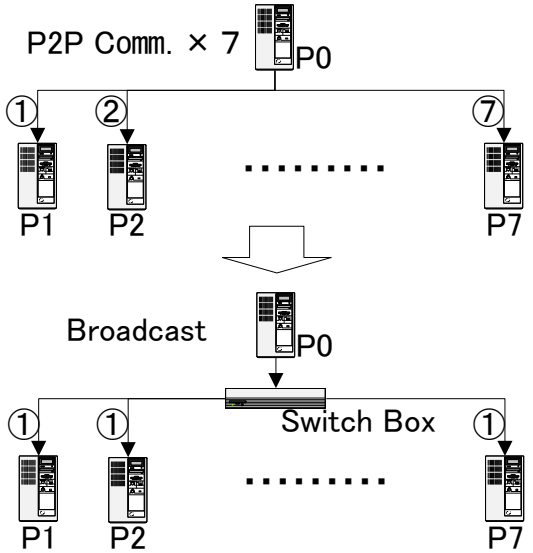

図 9 1 対全通信の例（8 台実行時）

5. 評　　価

5.1 評 価 環 境

本研究において実装した，クラスタ向けネットワークへのオフロードを行った DSM 実行時ライブラリの性能評価を行う．まず MMP については，評価用として通信プロトコル制御をホスト側で行う MMP-HC（Host-Controlled）版と，通信プロトコル制御を Maestro 側で行う MMP-OL（Offloaded）版を用いる．そして，さらに依存性解析処理のオフロードを行った MDO 版，ブロードキャスト (Broadcast=BC) を実装した MDO 版の各手法をそれぞれ比較する．各手法のオフロードの度合いをまとめると表 1 のようになる．

表 1 各手法のオフロードの度合い

	通信プロトコル制御	依存性解析	Broadcast 通信解析
MMP-HC	host	host	-
MMP-OL	NI	host	-
MDO	NI	host	-
MDO(BC)	NI	NI	NI

そして，本実験を行うにあたり，表 2 に示す評価環境を用意した．

表 2 評価環境

ホスト PC	Intel Xeon 2.8GHz PC3200 DDR-SDRAM 1GB
OS	Debian GNU/Linux 3.1
使用ホスト数	4 台/8 台
ネットワーク	Maestro2 cluster network

この実験環境上で，OpenMP で書かれた各種アプリケーションプログラムをコンパイルしたものを評価用アプリケーションとし，実行時間を計測する．

- ガウス法によって連立 1 次方程式の解を求める
- ヤコビ法によって連立 1 次方程式の解を求める
  - ガウス法／ヤコビ法で，各々4 種類の DSM 実行時ライブラリの関数をコンパイルによって付与，合計 8 種類を用意
- 問題サイズ n*n
  - ガウス法 ： n = 2,048, 4,096
  - ヤコビ法 ： n = 960, 1,920

また，以下の時間計測は基本的に特定プロセスにて行う．

5.2 総実行時間の短縮効果

各実装の性能比較を行うために総実行時間の比較を行う．図 10 と図 11 は MMP-HC の結果を 100 とした場合の MMP-OL 版，MDO 版，MDO(BC) 版結果の割合である．

図 10 と図 11 より，ガウス法，ヤコビ法のいずれの場合でも，本実装によって総実行時間を削減できていることが分かる．実行台数が多いほど総実行時間中に占める通信時間の割合が大きく，尚且つ同期処理や依存性解析処理も多くなるので，総実行時間の短縮率は大きくなっている．

以下の説明では，通信プロトコルのオフロードを **(1)**,

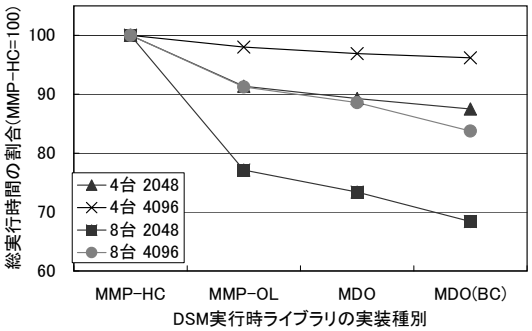

図 10 各手法におけるガウス法総実行時間の比率

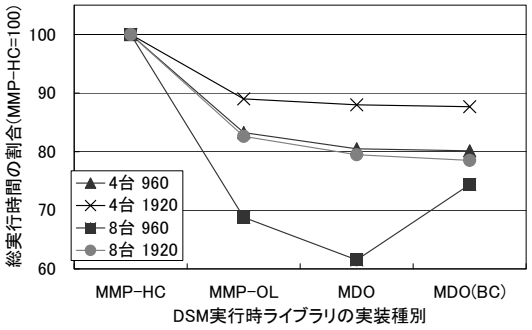

図 11 各手法におけるヤコビ法総実行時間の比率

依存性解析処理のオフロードを **(2)**，ブロードキャストの実装を **(3)** とする．図 10 よりガウス法では，**(1)** によって最大 23%， **(2)** によってさらに 3.7%，加えて **(3)** により 5%総実行時間を短縮できた．まとめると，MMP-HC 版から MDO(BC) 版までの実装により，合計で最大 32%と高い割合で総実行時間を短縮できた．

図 11 のヤコビ法の場合，8 台実行時の問題サイズ 960 において，**(1)** で各評価アプリケーション中最大の 31%，**(2)** でさらに 7.3%総実行時間を短縮できている．これは，ヤコビ法の依存データの通信パターンが全対全通信であるため，1 対全通信であるガウス法と比較して，総実行時間中に占める依存性解析処理や通信処理の割合が高く，通信プロトコルや依存性解析処理のオフロードが高い効果をあげているためと考えられる．しかし，**(3)** の実装を行うと，逆に 13%総実行時間が増大している．これは，ネットワークインタフェース側で行われる依存性解析処理時間とブロードキャスト通信パターン解析処理時間との合計が，ホスト側で行われるアプリケーション演算処理時間を上回ってしまい，ホスト側にて通信開始待ち (ネットワークインタフェースの処理完了待ち) 時間が発生したためである．また，現在は全対全通信を 1 対全通信の繰り返しで実装しているため，ガウス法と比べ集団通信部のオーバヘッドが大きいことも原因として挙げられる． この現象は，実行台数が多く，問題サイズが小さいという場合に発生しやすいが，図 11 の問題サイズ 1,920 の結果が示す通り，一般的な並列アプリケーションのように十分に大きな問題サイズを取ればこの現象は発生しない．

5.3 通信プロトコル処理のオフロードによる通信時間の短縮効果

MMP-OL による通信プロトコル処理のオフロード結果を評価する．図 12 は MMP-HC 版と MMP-OL 版の通信時間の比較を各アプリケーションにて行ったものである．

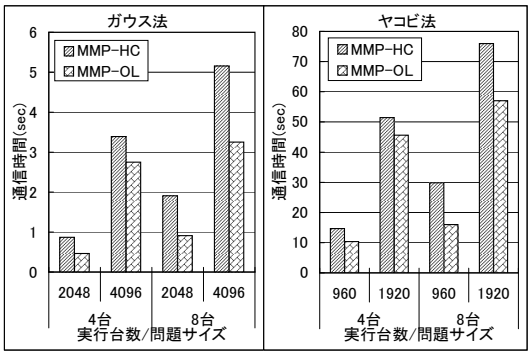

図 12 通信時間 (MMP-HC/MMP-OL)

図 12 より，ガウス法では最大 52%，ヤコビ法では最大 46%といずれの場合でも，通信プロトコルのオフロードを行った本実装により通信時間を大きく削減できていることが分かる．各アプリケーションに共通して，通信時間そのものが大きくなる 8 台実行時にて，絶対量としては大きく削減されている．一方，削減の割合を見ると，2 種類のアプリケーションの各問題サイズに共通して，30%〜40%程度削減できている．このことから，1 回の通信ごとに一定の割合で通信時間を削減できていることが分かる．

5.4 ブロードキャストによる通信時間の短縮効果

MDO 機構によりブロードキャストの実装を行った結果を評価する．図 13 は MDO 版と MDO(BC) 版の通信時間の比較を各アプリケーションにて行ったものである．

図 13 より，ガウス法では最大 54%，ヤコビ法では最大 27%，といずれの場合でも，MDO 機構によるブロードキャストの実装により通信時間を大きく削減できている．また，1 対 1 通信に基づいた従来の通信手法 (MDO 版) では，実行台数が増加するに従って通信

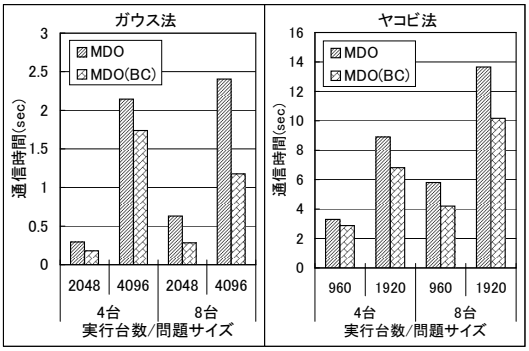

図 13 通信時間 (MDO/MDO-(BC))

時間も大幅に増加するのに対して，MDO 機構を用いたブロードキャスト (MDO(BC) 版) では，実行台数に関係なく 1 度の送信機会でブロードキャストが完了する．このことから，図 13 でも明らかなように，4 台実行時より 8 台実行時の方が，ヤコビ法では 10%程度，ガウス法では 20%程度，通信時間をより大きく削減できている．ここで，ヤコビ法と比較して，ガウス法はブロードキャストの効果がより高いことが評価結果より分かる．これは依存データの通信パターンが全対全通信であるヤコビ法では，依存データの授受を完了するには実行台数分のブロードキャストを繰り返す必要があることに対して，依存データの通信パターンが 1 対全通信であるガウス法では，1 回のブロードキャストで効率よく依存データの授受が完了するからである．

6. まとめと今後の課題

本稿では，クラスタ向けネットワークへのオフロードに基づいた OpenMP プログラムの高速実行方式の提案と，実装・評価を行った．従来の DSM 実行時ライブラリの実装上の問題点を示し，それを解決するために，本研究室で開発したクラスタ向け高性能ネットワークである Maestro cluster network に備わる MDO 機構を用いたオフロードに基づく実装方法を提案した．本実装によって，オフロードを用いない通信ライブラリである MMP-HC を用いた場合と比較して，通信プロトコル処理のオフロードによって 2%〜32%，依存性解析処理のオフロードでさらに 2%〜7.3%，加えてブロードキャストの実装で 2%〜6.2%実行時間を短縮できた．次に，各実装において短縮できたポイントに着目すると，通信プロトコル処理のオフロードを行った MMP-OL は MMP-HC と比較して通信時間を 15%〜53%短縮させることができた．そして，ブロードキャストを実装した MDO(BC) 版では，MDO 版と比較して通信時間を 15%〜61%短縮することができた．

今後の課題としては，さらに様々なアプリケーションを用いて評価を行うことで，本実装の有効性を検証すると共に，詳細な実行時間のプロファイルを解析し，依存性解析部や通信部の最適化を行うことが挙げられる．また，依存性解析処理の一部を Maestro に搭載されている FPGA 上で行うことを検討している．そして，Maestro 第 3 世代である Maestro3[2] 上で，本システムを完全に動作させることを予定している．これらの実装により，依存性解析処理時間や通信時間を短縮させ，前述したアプリケーションの問題サイズが小さい場合に通信開始待ち時間が発生する現象などを解決し，あらゆる並列アプリケーションに柔軟に対応可能な高性能 DSM の実現を目指す計画である．

謝辞 本研究の一部は，魅力ある大学院教育イニシアティブ「実践 IT 力を備えた高度情報学人材育成プログラム」による．

参 考 文 献

1) Aoki, K., Wada, K., Maruoka, H. and Ono, M.: Architecture and Performance of Dynamic Offloading Mechanism for Maestro2 Cluster Network, *Journal of the Information Processing Society of Japan(IPSJ-DC)*, Vol.3, pp.683–692 (2007).
2) Kuribayashi, H., Yasuda, K., Aoki, K., Wada, K. and Ono, M.: An Architecture and Performance of Maestro3 Cluster Network, *Proc. of IEEE Pacific Rim Conference on Communications*, Computers and Signal Processing, pp. 477–480 (2007).
3) P., P. S.: *Parallel Programming with MPI*, Morgan Kaufmann Publishers (1997).
4) 小倉崇浩: 分散環境における共有メモリ型並列プログラムの実行方式と評価, 修士論文, 筑波大学大学院システム情報工学研究科コンピュータサイエンス専攻 (2005).
5) Yamagiwa, S., Ferreira, K., Aoki, K., Ono, M., Wada, K. and Sousa, L.: *Maestro2*: Experimental evaluation of communication performance improvement techniques in the link layer, *Journal of Interconnection Networks*, Vol. 7, No. 2, pp. 295–318 (2006).
6) Yonezawa, N. and Wada, K.: *quad*: an Array Section Descriptor for Parallel Computing, *Proceedings of IASTED International Conference on Network, Parallel and Distributed Processing and Applications*, pp. 46–52 (2002).
7) Yonezawa, N., Wada, K. and Aida, T.: Barrier Elimination Based on Access Dependency Analysis for OpenMP, *Proc. of the*

2006 International Symposium on Parallel and Distributed Processing and Application (ISPA2006), Lecture Notes in Computer Science, Vol. 4330/2006, pp. 362–373 (2006).

8) Yonezawa, N., Wada, K. and Ogura, T.: *Quaver*: OpenMP Compiler for Clusters Based on Array Section Descriptor, *Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Networks*, pp. 234–239 (2005).

9) 青木圭一, 山際伸一, 和田耕一, 小野雅晃: Maestro2 クラスタネットワーク向けメッセージパッシングライブラリの開発と評価, 電子情報通信学会論文誌 D, Vol. J89-D, No. 5, pp. 919–931 (2006).