計算機アーキテクチャ 152－8
（２００３．３．１０）

圧縮を用いたリモートメモリ操作の効率化

青木 隆史† 堀 幸雄† 石橋 延夫††

本稿では情報圧縮を用いてクラスタ間で効率的なリモートメモリ操作を実現する逐次圧縮通信方式について提案する．逐次圧縮通信方式は情報圧縮処理とデータ通信処理を交互に切り替えることで実現する．圧縮処理における圧縮・伸長バッファサイズと通信処理における送信・受信バッファサイズを操作することで各処理速度に差があらわれることに着目し，逐次圧縮通信方式への適用を行った．逐次圧縮通信方式における各バッファサイズとスループットの関係を示し，これを評価する．また実アプリケーションでの性能を調べるために，並列ベンチマークテストのひとつである姫野ベンチマークを実行し効率化指標を測った．評価の結果から，逐次圧縮通信方式の有効性が示された．情報圧縮を用いることにより，ネットワークの物理的制限を超えて処理を行う方法について報告する．

An Efficient Remote Memory Operations with Data Compression

Takashi AOKI†, Yukio HORI† and Nobuo ISHIBASHI††

In this paper, we propose the Successive-Compression-and-Successive-Communication method which is a new approach to efficient implement of remote memory operation with data compression on cluster system. We produce the method by getting the data compression process and the data communication process to work in several shifts. We focused attention on differences of throughput speed brought about operating compression buffer on the data compression process and operating communication buffer on the data communication process, and we applied them to the method. We show relations between each buffers and throughput speed, and evaluate them. We executed the Himeno Benchmark to survey a system performance with our method, and measured barometer of efficiency. The results show an efficiency of using our method. We report a way of processing the network communication using data compression over physical restriciton.

1 はじめに

並列計算機の効率向上を目的としてネットワーク・トポロジ，制御方式 (SIMD (Single Instruction Maltiple Data stream) あるいは MIMD (Multiple Instruction / Multiple Data))，動作モード (同期式あるいは非同期式) 等の観点から幅広く研究が行なわれている．ネットワークを介したプロセス全体の計算コストは代数的演算のコスト，ファイル I / O のコスト，プロセッサ間通信のコストにわけて考えられ，その中でも通信のコストは並列計算機の効率を大きく左右するものと考えられている [1]．

本稿では広範囲な情報を効率的に圧縮することで並列計算機の効率を向上させる手法について述べる．転送データの逐次圧縮を行うことによりネットワークにおけるデータの衝突を軽減させ，転送媒体の物理限界よりも多くのデータを転送できる手法を提案する．データ転送におけるインターフェイスには MPI (Message Passing Interface) [8][9] を，圧縮・伸長のためのライブラリには zlib [2][3] を使用した．zlib の利用により情報が可逆圧縮されて転送されるため，送信ノードと受信ノードにおいて圧縮前情報の整合性を保障したうえで通信性能の効率化を図るものである．

さらにこの手法を並列ベンチマークテストに適用した場合の性能を検証し，この手法が並列計算時におけるプロセッサ間の通信効率をどの程度向上し，プロセス全体の計算コストをどの程度削減させられるかを定量的に評価する．

また本稿で述べる手法はソフトウェアレベルによる制御であり，MPI を用いたデータ通信処理を行うプログラムへの組み込みは非常に容易である．同時により安価な計算機における実現も目指しており，特殊な計算機でないと功を奏さない類のものではなく，計算機を所有する誰もが実装を可能とする手法である．

†神奈川大学大学院理学研究科情報科学専攻
Department of Information Science, Faculty of Science, Kanagawa University
††神奈川大学理学部情報科学科
Department of Information Science, Faculty of Science, Kanagawa University

2 逐次圧縮通信方式

データ転送処理と圧縮処理の組み合わせとして次のふたつの手法が考えられる．

圧縮してから通信法　送信側では送信すべき全ての情報を圧縮し，圧縮処理が完了した後に既圧縮データを通信メッセージとして送信する．受信側では圧縮されたデータを通信メッセージとして全て受信し終えた後，伸長処理を実行し圧縮前の情報を取り出す．

圧縮しながら通信法　圧縮や通信の出力をパイプライン的に処理する方法である．送信側では送信すべき全ての情報を圧縮し終えていなくても，圧縮されたデータがいくらかでも出力されていたらそれを通信メッセージとして送信する．メッセージ送信後は元の情報の続きを圧縮するというように，圧縮処理と送信処理を交互に繰り返す．こうすることによって最終的に圧縮された情報を全て送信する．受信側では圧縮されたデータを通信メッセージとして受信する度に伸長処理を行い，伸長処理の出力結果を次々につなぎあわせることで最終的に圧縮前の情報を取り出す．

圧縮と通信というふたつの処理を行う場合，「してから法」では全情報の圧縮が終了しないとメッセージ送信を行うことができないというように，一方の処理が完全に終了しないと他方が処理を開始できない．メッセージ送信中に次の情報を圧縮しようにも圧縮すべき情報が既に存在しない状態である．したがってプログラム実効時間は単純に圧縮処理時間 + 送信処理時間，あるいは受信処理時間 + 伸長処理時間で計算される．

これに対し「しながら法」は情報圧縮中にそれまでに圧縮されたデータをメッセージとして送信するという考え方で，ふたつの処理のオーバーラップを可能にすることができる．この場合圧縮処理時間と送信処理時間あるいは受信処理時間と伸長処理時間の一部が重なることとなり，プログラム実効時間を短縮し作業の効率化を図る余地があると推測できる．

我々はここに述べた「しながら法」の考え方を元にデータの逐次圧縮通信方式を実現する．圧縮 (伸長) 処理と送信 (受信) 処理のオーバーラップを行うことを考えた場合，圧縮 (伸長) 関数と送信 (受信) 関数にノンブロッキング関数の使用が要求される．そのためメッセージ・パッシングにおいては MPI のノンブロッキング送受信関数 (MPI_Isend()，MPI_Irecv()) を用いる．しかし zlib にノンブロッキング関数は存在しない．そこで圧縮・伸長関数には圧縮中または伸長中にバッファ操作を可能とする関数 (deflate()，inflate()) を選択する．

2.1 圧縮送信処理

zlib の圧縮関数 deflate() は可能な限りの量のデータを圧縮し，入力バッファが空になるか出力バッファがいっぱいになった時点で処理を止める[1]．このときユーザは入力バッファに情報の続きを読み込ませるか，出力バッファのデータを退避して出力バッファを再利用可能にさせる必要がある．この「出力バッファを再利用可能にさせる」瞬間に現時点までの圧縮データを送信する機会が生まれる．このときに deflate() の出力バッファを MPI_Isend() の送信バッファとして指定し，メッセージ送信処理を行う．

MPI のノンブロッキング送信関数 MPI_Isend() による通信を完了させるには MPI_Wait() 等を用いる．MPI_Wait() は送信バッファが利用可能[2] となるまでブロックするため，MPI_Isend() 呼び出し直後から MPI_Wait() 呼び出し直前までがオーバーラップを可能とするタイミングである．したがって 1 回目の deflate() 実行後直ちに MPI_Isend() でメッセージを送信し，この送信処理に対応する MPI_Wait() 発行までに 2 回目の deflate() を実行する．この動作を繰り返すことにより逐次圧縮通信の圧縮送信処理を構成する．

ここで，deflate() の圧縮バッファの大きさと MPI_Isend() の送信バッファの大きさをどの程度にするのが望ましいかという問題がある．deflate() にしろ MPI_Isend() にしろユーザにより指定されるバッファは任意のため，圧縮送信処理はふた通り設ける必要がある．図 1 は実装した処理の簡単な手順である．各々のバッファサイズにより処理をわけて実験を行う．

2.2 受信伸長処理

zlib の伸長関数 inflate() は可能な限りの量のデータを伸長し，入力バッファが空になるか出力バッファが一杯になった時点で処理を止める[3]．このときユーザが入力あるいは出力バッファの情報を更新する手続は deflate() の場合と同様である．

[1]ユーザは前もって非圧縮情報が格納されたバッファを入力バッファとして指定し，当該情報圧縮後のデータを格納したいバッファを出力バッファとして指定しておく必要がある．

[2]送信の場合は送信バッファ中のメッセージが通信サブシステムによって既にバッファリングされたか送信されて書き込み可，受信の場合は受信バッファが受信メッセージを取り込んでいるという意味である．

[3]deflate() の場合とは逆に既圧縮データが格納されたバッファを入力バッファとして指定し，伸長後の元の情報を格納したいバッファを出力バッファとして指定しておく必要がある．

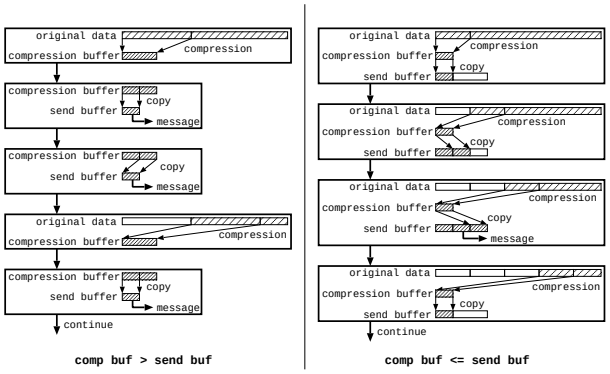

図 1: 圧縮送信処理手順

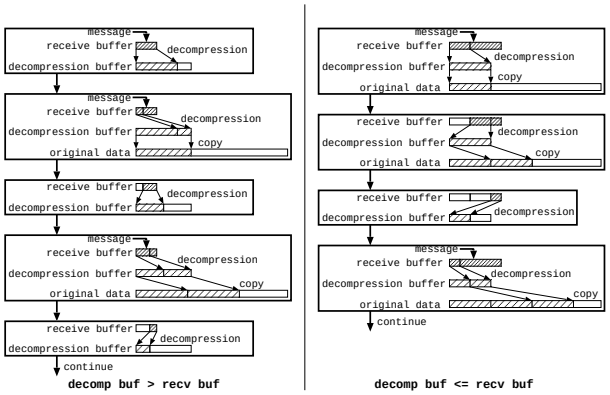

図 2: 受信伸長処理手順

MPI のノンブロッキング受信関数 MPI_Irecv() を完了させるのに MPI_Wait() を用いることは MPI_Isend() の場合と同様であり，MPI_Irecv() 呼び出し直後から MPI_Wait() 呼び出し直前までがオーバーラップを可能とするタイミングである．1 回目の受信処理が終了したら続けて 2 回目の受信処理を開始する．ここでは MPI_Irecv() に対応する MPI_Wait() を発行せずに，先程受信した 1 回目の既圧縮データを指定して inflate() を実行する．inflate() 終了後に MPI_Wait() を発行して 2 つ目の既圧縮データを受信完了する．この動作を繰り返すことにより逐次圧縮通信の受信伸長処理を構成する．

圧縮送信処理と同様に受信バッファの大きさと伸長バッファの大きさの問題があるため，受信伸長処理をふた通り設ける．受信伸長処理に実装した手順の概要を図 2 に示す．

3 実験と評価

情報源となる圧縮前のデータは乱数とし，入力サイズは桁数を増やしていくことで実験を行った．計算機の環境は表 1，通信機器の環境は 表 2 の通りである．通信処理が発生する実験においては当該計算機を 2 台使用することで処理能力に偏りが生じる可能性を排除した．通信時間の計測は Round-trip で行い，かつ時間計測関数の誤差を小さくするために少なくとも数秒間の測定を行うように同じ通信処理を繰り返した．数秒間の通信処理が終了した後に経過時間を繰り返し回数で除算し，1 回の通信処理に費された時間を計算した．

表 1: 評価実験計算機環境

OS	FreeBSD 4.6.2R
CPU	Pentium III Processer 2.0 GHz × 2
Memory	1.0 GB

表 2: 評価実験通信環境

Network interface	100 BASE-TX
Switching HUB	100 BASE-TX
	800 Mbps (全二重) Trunk

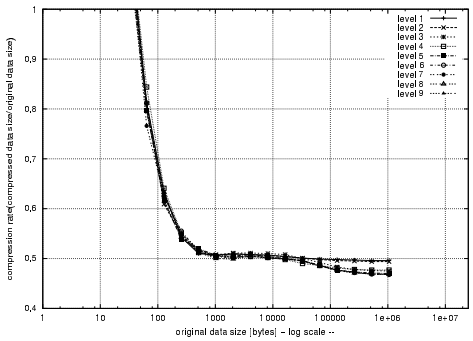

図 3: 圧縮レベルごとに表示した入力サイズに対する圧縮率

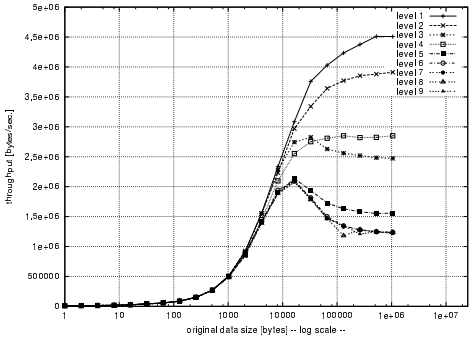

図 4: 圧縮レベルごとに表示した入力サイズに対する圧縮処理のスループット

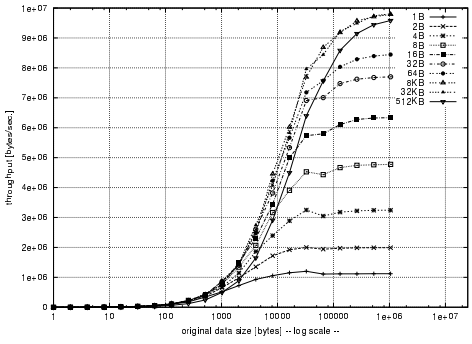

図 5: 圧縮バッファサイズごとに表示した入力サイズに対する圧縮処理のスループット

またこれら 2 台の計算機で構築するネットワーク・トポロジは，HUB を用いたスター型に接続されている．

3.1 圧縮，通信のバッファ制御とその評価

ここでは圧縮処理，通信処理それぞれ単体での処理性能を測定する．zlib の圧縮・伸長関数と MPI の送信・受信関数におけるバッファサイズを操作することにより，入力データのサイズと各処理におけるスループットの関係を示し，圧縮と通信の性能がそれぞれどのように違ってくるかを評価する．

3.1.1 zlib の圧縮レベルと圧縮率

zlib の圧縮関数は圧縮レベルの指定が可能であり，指定したレベルによって圧縮率と共に圧縮処理に費す時間が変化する．ここでは圧縮前のデータ長に対する圧縮後のデータ長の割合と，圧縮・伸長処理のスループットを圧縮レベルごとに評価する．また圧縮・伸長バッファの大きさを変化させたときのスループットをバッファサイズごとに評価する．

図 3 は 1 Byte から 1 MByte の乱数データに対する圧縮率を圧縮レベルごとに表示したグラフで，圧縮率は圧縮後のデータ長を圧縮前のデータ長で除算した値である．

図 3 からは入力サイズが 64 Byte 付近になって初めて圧縮後のサイズが入力サイズよりも小さくなっていることと，入力サイズが 1 KByte 付近になって初めて最大圧縮率に近い性能を発揮することを見てとることができる．

また今回実験した乱数データの場合ではいずれの圧縮レベルのグラフも圧縮率が 0.5 付近 (入力サイズに対しおよそ半分) で収束する．したがって zlib の圧縮レベルオプションを指定することによる圧縮率の違いはほとんど期待できないと言える．

図 4 は 1 Byte から 1 MByte の乱数データに対する圧縮処理のスループットを圧縮レベルごとに表示したグラフで，圧縮処理のスループットは入力サイズを圧縮処理時間で除算した値である．

どのグラフの勾配も収束に向かう方向に傾いているため，入力サイズをこれ以上増やしても値はそれ程変化しないと推測できる．またレベル 1 とレベル 9 では 4 倍程度の差があることから，入力サイズが大きい場合には圧縮レベルを小さく設定した方が単位時間あたりに圧縮可能な情報量が多いということがわかる．

図 3 に見られる圧縮率の違いは小さいものであったことから，総合的な圧縮性能に着目したとき，圧縮レベルをレベル 1 に設定して圧縮処理を行うことが望ましいと言うことができる．

これらの結果から圧縮レベルをレベル 1 に限定して inflate()，deflate() に圧縮バッファ操作を適用した．図 5 は 1 Byte から 1 MByte の乱数データに対する圧縮処理のスループットを，圧縮バッファサイズごとに表示したグラフである．ただし全てのグラフを描くと読み取りにくくなるため，いくつかの圧縮バッファサイズに関する測定結果を省略した．圧縮処理のスループットは図 4 と同様に入力サイズを圧縮処理時間で除算した値である．

図 5 から圧縮・伸長のためのバッファサイズを操作することにより圧縮処理のスループットが大きく異なってくることがわかる．したがって図 5 からは，入力サイズが大きいときは圧縮バッファもそれなりに大きくなければならな

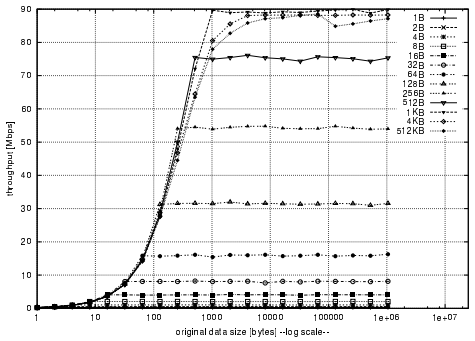

図 6: 通信バッファサイズごとに表示した入力サイズに対する通信処理のスループット

表 3: スループットの差

入力サイズ	バッファサイズ	スループットの差
128 [KB]	64 [KB]	+ 2.67 [Mbps]
256	64	+ 1.78
512	64	+ 2.10
1024	64	+ 1.67

いことが推測できる．

3.1.2 MPI 通信のスループット

MPI を用いたデータ通信処理において，通信バッファの大きさを変化させたときのスループットをバッファサイズごとに評価する．

図 6 は 1 Byte から 1 MByte の乱数データに対する通信処理のスループットを，通信バッファサイズごとに表示したグラフである．ただし図 5 と同じ理由によりいくつかの通信バッファサイズに関する測定結果を省略した．通信処理のスループットは入力サイズを通信処理時間で除算した値である．

通信バッファの大きさが 1 KByte 以下のとき，入力サイズに等しい通信バッファを確保した場合をピークに，以降入力サイズが大きくなってもスループットは横這いとなる．また通信バッファの大きさごとにピークの値が異なり，確保した通信バッファが小さい程スループットは小さい値で収束する．

しかし通信バッファの大きさが 1 KByte を超えたときは入力サイズが 1 KByte 強の場合にスループットが収束し始め，各通信バッファサイズは通信環境の性能の理論値[4]に対して一律 90 % に近いスループットを記録している．バッファサイズを 1 KByte に設定して通信処理を行った場合に最もスループットが良く，そのスループットはおよそ 90 Mbps であった．

入力サイズと通信バッファサイズが同じ大きさのときは，バッファ操作なしで通信処理を行うことと等しい．本節の実験で入力サイズとは異なる大きさの通信バッファサイズを用いて通信を行ったときに，バッファ操作なしのパターンとの差が顕著であったものを表 3 にまとめる．

この結果 MPI を用いた通信処理において通信バッファ操作を施すことは，僅かではあるがスループットを向上させると考えられる．

[4]表 2 を参照．

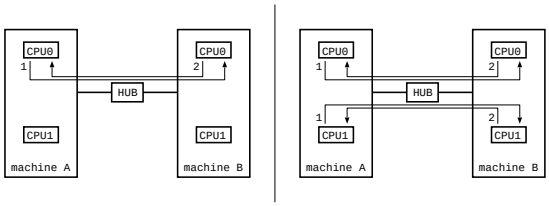

図 7: 実験における各システムの構成

3.2 逐次圧縮通信方式の評価

zlib と MPI がそれぞれ提供する圧縮と通信の関数におけるバッファ操作により，各処理のスループットが変化するのは 3.1 節で示した通りである．ここでは逐次圧縮通信の手順を用いてスループットを測定する．逐次圧縮通信にバッファ操作を適用し，スループットが最良となる圧縮バッファと通信バッファの組み合わせをシステマティックに調査，評価することが目的である．通信処理における通信時間計測方法は 3 章の冒頭で述べた通りである．

使用した計算機は 2 台 でそれぞれがデュアルプロセッサマシンのため，次のふた通りのシステムにおける調査を試みた．

1. 2 CPU (1 CPU × 2 台) を用いたシステム
2. 4 CPU (2 CPU × 2 台) を用いたシステム

各システムの構成図を図 7 に示す．図 7 は左側が 2 CPU を用いたシステムで，右側が 4 CPU を用いたシステムである．

計算機の名称を A，B とし A，B それぞれの CPU を 0，1 とした．以降，ある特定のプロセッサのことをいう場合には計算機の名称と括弧書きの番号で表記する[5]．また矢印はひとつでメッセージの送信処理と受信処理とをあらわし，矢印の始点におけるプロセッサからメッセージを送信して矢印の終点におけるプロセッサでメッセージを受信することを意味する．そしてその脇に書かれた数字はメッセージの送受信処理を行う順序である．4 CPU を用いたシステムにおいては A(0) から B(0) に向けた通信処理と A(1) から B(1) に向けた通信処理が同期をとって開始される．

図 7 に示したふた通りのシステムにおいて調査を行い，それと同時に非圧縮通信との性能とも比較し，併せて評価を行った．ここでスループットは 1 CPU あたりの値であり，情報源となる乱数データ長は一律 512 KByte に設定するものである．

3.2.1 2 CPU を用いたシステムでの調査

図 8 は計算機 2 台がそれぞれ 1 CPU を使用した場合における 1 CPU あたりの圧縮通信処理のスループットである．通信バッファサイズに対するスループットを圧縮バッファサイズごとに表示したものである．圧縮バッファサイズと通信バッファサイズは共に 128 Byte から 512 KByte まで変化させた．ただしグラフを読みやすくするためにいくつかの圧縮バッファサイズに関する測定結果を省略した．また比較のために通信バッファサイズのみ変化させた非圧縮通信時のスループットも併せて表示した．

[5]例えば計算機名称 A，CPU 番号 1 のプロセッサをあらわしたい場合には，A(1) という記号で表記する．

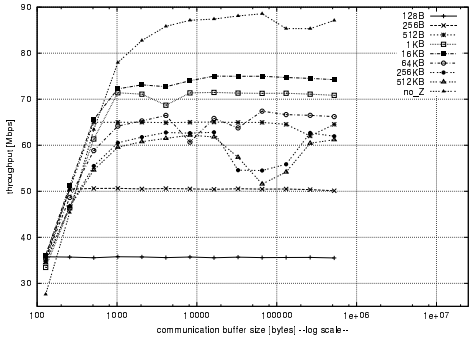

図 8: 圧縮バッファサイズごとに表示した 1 CPU あたりの逐次圧縮通信処理のスループット (2 CPU)

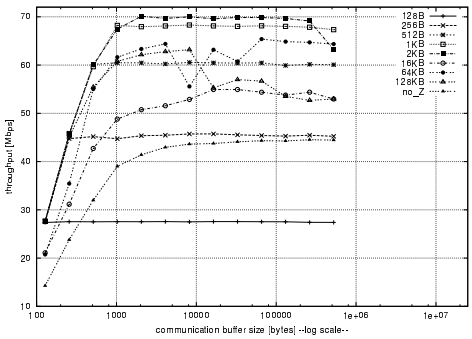

図 9: 圧縮バッファサイズごとに表示した 1 CPU あたりの逐次圧縮通信処理のスループット (4 CPU)

表 4: 最短通信時間 (2 CPU)

	逐次圧縮通信	非圧縮通信
スループット [Mbps]	75.02	88.56
最短通信時間 [sec]	0.107	0.090
圧縮バッファ [KB]	16	–
通信バッファ [KB]	16	64

表 5: 最短通信時間 (4 CPU)

	逐次圧縮通信	非圧縮通信
スループット [Mbps]	70.12	44.53
最短通信時間 [sec]	0.114	0.180
圧縮バッファ [KB]	2	–
通信バッファ [KB]	2	256

通信環境の性能の理論値に達することはないにせよ，非圧縮通信が 90 Mbps に近いスループットを示したのに対して逐次圧縮通信が示したスループットは最高でおよそ 75 Mbps であった．そしてそのときの圧縮バッファサイズと通信バッファサイズは共に 16 KByte であった．

圧縮処理により元の情報はおよそ半分になるが，圧縮処理に費される時間が長いため元の情報を非圧縮で通信した方が通信効率が良い．逐次圧縮通信時と非圧縮通信時それぞれの最短通信時間を表 4 に示す．この時間は入力サイズ 512 KByte のデータに対する各通信の Round-trip time である．

3.1.1 節の図 5 に示した結果によると入力サイズが 512 KByte，圧縮バッファサイズが 32 KByte のときにスループットが最良であった．同様に 3.1.2 節の図 6 に示した結果によると入力サイズが 512 KByte，通信バッファサイズが 1 KByte のときにスループットが最良であった．このことから本節の結果は必ずしも図 5 と図 6 にしたがった結果にはならないケースであった．

3.2.2 4 CPU を用いたシステムでの調査

図 9 は計算機 2 台がそれぞれ 2 CPU を使用した場合における 1 CPU あたりの圧縮通信処理のスループットである．通信バッファサイズに対するスループットを圧縮バッファサイズごとに表示したものである．ただし図 8 のときと同じ理由からいくつかの測定結果は省略した．

ひとつのネットワークインターフェイスに対してふたつの CPU が絶え間なく通信要求を開始するため，非圧縮通信時のスループットは計算機 1 台に対して 90 Mbps となり，1 CPU あたりのスループットはその 50% の 45 Mbps であった．これに対し逐次圧縮通信における 1 CPU あたりのスループットは約 70 Mbps で，そのときの圧縮バッファサイズと通信バッファサイズは共に 2 KByte であった．この 70 Mbps というスループットは計算機 1 台に対して 140 Mbps の計算になり，通信媒体の物理的制限を超えた性能を示す結果となった．

逐次圧縮通信時と非圧縮通信時それぞれの最短通信時間を 3.2.2 節と同様に Round-trip time で表 5 に示す．

通信媒体上を流れる転送データが圧縮処理によっておよそ半分になっていることと通信処理の隙間に圧縮処理が入ることによって，非圧縮通信時と比べて転送データの衝突が起こりにくくなっているものと考えられる．

3.3 並列ベンチマークへの適用

ここでは前節までの手法について姫野ベンチマーク[6] を用いて逐次圧縮通信方式を評価する．姫野ベンチマークはポアソン方程式解法をヤコビの反復法で解く場合に，主要なループの処理速度を計るものである．

また姫野ベンチマークにおけるリモートメモリ操作の出現回数 R は以下の式で表わすことができる．なお，x, y, z は問題サイズ (表 6) のそれぞれの分割数を表わしている．これよりノード数が増えるに従いリモートメモリ操作が並列計算機の効率を大きく左右するものと推測できる．

$$(\acute{x}, \acute{y}, \acute{z}) = \begin{cases} 1 & (\text{if } x, y, z \geq 2) \\ 0 & (\text{if } x, y, z = 1) \end{cases}$$

$$R = (4\acute{x} + 4\acute{y} + 4\acute{z}) + 2xyz$$

前節までの環境において，逐次圧縮通信方式と非圧縮通信方式を用いた場合のそれぞれの結果を図 10 に示す．問題サイズは M で実行した．どちらもノード数が増えると並列化効率が下がり性能向上が低くなるが，本手法を用いることでどのケースの場合についても元の並列化効率より 10 % 程度の改善が見られた．

[6] http://w3cic.riken.go.jp/HPC/HimenoBMT/index.html

表 6: 姫野ベンチマークの問題サイズ

問題サイズ	配列サイズ
XS	64 x 32 x 32
S	128 x 64 x 64
M	256 x 128 x 128
L	512 x 256 x 256

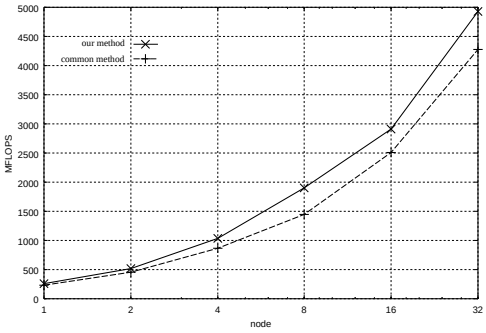

図 10: 姫野ベンチマークによる評価結果

4 おわりに

zlib を用いた圧縮処理における圧縮バッファ操作の効果を測定し，圧縮処理のスループットに差があらわれることを示した．また MPI を用いた通信処理における通信バッファ操作の効果を測定し，通信処理のスループットに差があらわれることを示した．

そしてこれらを複合させ，逐次的に圧縮して逐次的に通信を行う逐次圧縮通信方式を提案し計算機上に実装した．

100 BASE-TX でスター型に接続されたデュアルプロセッサマシン 2 台を実験環境とし，逐次圧縮通信方式を適用した通信スループットをシステマティックなバッファ操作を用いて測定した．同一環境で非圧縮通信の基本性能を測定した結果，4 CPU を用いた実験において，基本性能 90 Mbps に対し約 1.5 倍の 140 Mbps を達成した．これは理論上通信媒体の限界を超える通信スループットを示す結果となった．

さらに逐次圧縮通信方式を並列アプリケーションである姫野ベンチマークへ適用したところ，非圧縮通信時の並列化効率を 10 % 程度改善する結果となった．

本稿で提案した逐次圧縮通信方式はプロセス全体の通信コストを削減させ，クラスタリングにおける並列化効率を向上させることの意味において有効性を示すものである．

また本手法はソフトウェアレベルの制御で実現され，圧縮処理のために多少の CPU パワーは要求されるものの，専用のハードウェアを導入することなく一般の計算機に実装が可能であり，実装の容易性という観点から見ても有用であると言える．

参考文献

[1] Gentleman, W.M.: Some Complexity Results for Matrix Computations on Parallel Processors, *J. ACM*, Vol. 25, No. 1 pp. 112-115, 1978.

[2] The gzip home page: `http://www.gzip.org/`

[3] A Massively Spiffy Yet Delicately Unobtrusive Compression Library: `http://www.gzip.org/zlib/`

[4] Jacob Ziv, Abraham Lempel: A universal algorithm for sequential data compression: IEEE Transactions on Information Theory, Vol. 23, No. 3, pp. 337-343, May. 1977.

[5] James A. Storer and Thomas G. Szymanski: Data compression via textual substitution, Journal of the ACM, Vol. 29, No. 4, pp. 928-951, Oct. 1982.

[6] 福島荘之介: ファイル圧縮ツール gzip のアルゴリズム, 共立出版 bit, Vol. 28, No. 3, pp. 30-37, Mar. 1996.

[7] 韓太瞬, 小林欣吾: 情報と符号化の数理, 培風館, 1999.

[8] The Message Passing Interface(MPI) standard: `http://www-unix.mcs.anl.gov/mpi/`

[9] MPICH - A Portable Implementation of MPI: `http://www-unix.mcs.anl.gov/mpi/mpich/`

[10] Wong, F. C. and Culler, D. E.: Message Passing Interface Implementation on Active Messages. `http://now.CS.Berkeley.EDU/Fastcomm/MPI/`

[11] Message Passing Interface Forum: *MPI-2: Extensions to the Message-Passing Interface* (1997). `http://www.mpi-forum.org/docs/mpi-20-html/mpi2-report.html`

[12] 建部修見, 児玉祐悦, 関口智嗣, 山口喜教: リモートメモリ書き込みを用いた MPI の効率的実装, 情報処理学会論文誌, Vol. 40, No. 5, pp. 2246-2255(1999).

[13] S. Arramreddy, D. Har, K. Mak, T.B. Smith, R.B. Tremaine, and M. Wazlowski: IBM Memory eXpansion Technology (MXT) Debuts in a ServerWorks Northbridge, submitted for publication in IEEEmicro, October 13, 2000.

[14] MXT for Linux: `http://www-124.ibm.com/mxt/`