OpenMPI における集団通信のオーバヘッドの分析

2010SE026 福井優介　2010SE064 石井隆義
指導教員：宮澤元

1　はじめに

並列計算の手法の 1 つとして，ネットワークで接続された複数の計算機を用いて並列処理を行う分散並列計算が広く利用されている．分散並列計算では，各計算機上で動作するプロセス同士がメッセージを通信し，並列計算を行う．特に，大規模な並列計算を行う際には，Message Passing Interface (MPI)[6] と呼ばれる並列計算ライブラリが広く使われている．

CPU による計算のコストに比べて，ネットワーク上のメッセージ通信のコストは大きいので，MPI アプリケーションの性能を向上するためには，メッセージ通信のコストを抑えることが必要である．そこで，MPI におけるメッセージ通信のコストを下げるために様々な工夫がなされている．メッセージ通信の局所性を利用して MPI プロセスを配置することもその一例である [4]．また，メッセージ通信を行なっている間に別の計算をおこなってメッセージ通信のコストを隠蔽するようなアルゴリズムを用いるといった工夫もある．

しかし，近年，CPU の性能向上が著しく，ネットワークで接続された複数の計算機を用いて並列処理をする際のメッセージ通信のコストが相対的に増加しており，メッセージ通信コストを抑えることが困難となってきている．我々の予備実験の結果からは，集団通信のデータ量が少ないとき，ホスト数を増やしても性能向上しないケースがあることがわかっている．

本研究では，分散並列計算用に広く利用されている OpenMPI[6] ライブラリを用いて通信オーバヘッドの分析を行う．特に集団通信の MPI_Allreduce() と MPI_Alltoall() に着目し様々な条件における通信オーバヘッドを測定する．

2　研究の背景

MPI について，既存の MPI システムを紹介すると共に本研究で着目する MPI のいくつかの集団通信のアルゴリズムについても示す．また，本研究の動機についても述べる．

2.1　既存の MPI システム

OpenMPI

OpenMPI は，様々な先行プロジェクトの成果をもとに開発されている MPI 実装である．これらの個々のプロジェクトからの優れたアイディアと技術を使用し，1 つのまとまったクラスの作成をすることで様々な分野で優れた MPI の実装を目指している．

MPICH

MPICH[6] は，高性能かつ自由に利用できる MPI 標準のポータブルな実装であり，最も多く利用されているものの 1 つである．MPICH は MPI-1 規格[*1]を実装しているが，既に開発が終了しており，メンテナンスも公式には行われていない．

2.2　MPI 集団通信のアルゴリズム

本節では，MPI 集団通信のうち MPI_Allreduce() と MPI_Alltoall() についてそのアルゴリズムを示す．

2.2.1　MPI_Allreduce() のアルゴリズム

MPI_Allreduce() は，全プロセスのデータを対象に演算を行い，結果を全プロセスへ送信する [2, 7, 8]．Allreduce 操作において，p を参加プロセス数とすると全ステップを終えるには少なくとも $2(p-1)$ 回の操作が必要である．

Allreduce を実行する様々なアルゴリズムが実装されている [5]．Basic Linear アルゴリズムは MPI_Allreduce() 実装で最も基本的な部分であり，OpenMPI において fall-back として利用されている．Recursive Doubling はステップ n で i 番目のプロセスが 2^n だけ離れたプロセスとメッセージを交換するアルゴリズムである．各ステップごとにプロセスが送受信するデータを倍にして通信をする．プロセス数が “power-of-two(2^n の場合)” と “non-power-of-two(2^n ではない場合)” に分けられている．Power-of-two はプロセス数が 8 のとき，まず隣同士で交換し，次に 2 つ離れたランク同士で交換し，最後に 4 つ離れたプロセス同士で交換する．通信のステップ数は $\log p$ である．一方，non-power-of-two は Rabenseifner のアルゴリズム [3] を元に作られたアルゴリズムである．Recursive Doubling の形の通信を行うために，$[\log p]$ を超過する数のプロセスが余計に通信を行う必要がある．よって，この場合の通信ステップ数は $2[\log p]$ となる．ring は，i 番目のプロセスが各ステップで $i+1$ 番目のプロセスに対してメッセージを送信して，$i-1$ 番目のプロセスからメッセージを受信することを連続してリングのように通信するアルゴリズムである．OpenMPI は標準ではこれらのアルゴリズムを条件に応じて切り替えながら動作している．

2.2.2　MPI_Alltoall() のアルゴリズム

MPI_Alltoall() は，全プロセスが，全プロセスに対してスキャタを行うのと同じ操作が得られる集団通信である [2, 7, 8].

- Pairwise アルゴリズム [5] について

[*1] MPI には MPI-1 規格と MPI-2 規格が存在している．

常に 2 つのプロセスが通信ペアを組んで通信を行い，実行ステップごとにプロセスの組み合わせを変えていくアルゴリズムである．全てのプロセスがペアを組むので，全てのステップを終えるには $p-1$ ステップが必要である (p は参加する全プロセス数)．したがって，全てのプロセスと通信を行うためには，参加する全プロセス数が 2^n でなければならない．

- Modified Bruck アルゴリズム [5] について
  複数の受信プロセスに対するデータをまとめて送信することで，実行ステップの減少を図ったアルゴリズムである．i ステップにおいて $+i$ 離れたプロセスへ送信し，$-i$ 離れたプロセスから受信する．各ステップで，$p/2$ 倍のデータを送信しているので全てのステップを終えるには $\log_2 p$ ステップ必要である．また，全体では $\log_2 p$ ステップが必要であるが，プロセス数が多くなるにつれて 1 回の送受信データ量が増加するために，全体での総通信量は大幅に増加する．

2.3 本研究の動機

OpenMPI を用いて並列ダイクストラ法を実装し，最短経路問題の実行時間を調べたところ，計算機の台数に応じて実行時間が削減されないことがわかった．

この原因として，現在のコンピュータシステムにおいて CPU の性能向上やネットワークバンド幅の増加により，通信コストが計算コストに比べて増加しているのではないかと考えられる．特に，並列ダイクストラ法では，1 回の通信データ量が小さく，OpenMPI の最適化が働いていない可能性がある．

3 集団通信オーバヘッドの分析方針

2.3 節の実験に用いた並列ダイクストラ法 [1] のアルゴリズムでは，MPI_Allreduce() と MPI_Alltoall() などの集団通信を利用している．そこで，OpenMPI の集団通信の通信オーバヘッドを分析することとした．

並列ダイクストラ法では，1 回の通信サイズが小さいのに対し，MPI を用いた典型的なアプリケーションでは通信サイズが大きいことが多い．そこで，通信サイズを様々に変えて性能を測定する．また，OpenMPI の集団通信は，それぞれ何種類かのアルゴリズムが実装されており，切り替えて実行することができる．アルゴリズムによる通信コストの違いを見るために，アルゴリズムを切り替えて性能を測定する．

4 実験

3 章の方針に基づき実験を行った．まず，目的の集団通信のみを行う簡単なプログラムを用いてプログラムの実行時間を測定し，次に最短経路問題のプログラムを用いて同様の測定を行った．

4.1 実験環境

実験環境を表 1 に示す．表のサーバを 2〜8 台使用した．

	サーバ 8 台
CPU	Intel(R) Core(TM) i7-3770
クロック周波数	3.40GHz
メモリ	8GB
コア数	4core 8Threads
OS	Ubuntu 12.10
MPI	OpenMPI バージョン 1.6.3
ネットワーク	1000Base-T ギガビットイーサネット

表 1 コンピュータ及びネットワークの仕様

4.2 簡単なプログラムでのコスト測定

4.2.1 MPI_Allreduce()

MPI_Allreduce() 関数を用いて int 型の整数を演算し通信した．データサイズを変更しながら計測した．MPI_Allreduce() 関数を 1 万回ループさせ実行にかかった時間を測った．各グラフの縦軸は実行時間 (秒) を，横軸はプロセス数を，折れ線部分はアルゴリズム 0 から 5 までを示している．対応するアルゴリズムの名称を表 2 に示す．

AL0	default
AL1	Basic Linear
AL2	Reduce + Bcast
AL3	Recursive Doubling
AL4	Ring
AL5	Segmented Ring

表 2 MPI_Allreuce() アルゴリズム対応表

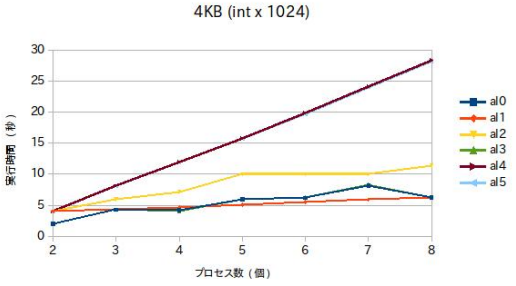

図 1 MPI_Allreduce() 4KB

図 1 にデータ量が 4KB の時の測定結果を示す．AL4，5 の時，実行時間はプロセス数に比例している．またプロセス数に関係なく最も遅い．プロセス数 5，6，7 のときアルゴリズムの切り替えにより多少速くなることがわかる．

図 2 にデータ量が 16KB の時の測定結果を示す．AL1 の時，実行時間はプロセス数に比例している．プロセス数に関係なく AL3 が最も速い．AL0 がかなり遅いことから切り替えが上手くいっていないことがわかる．

図 2　MPI_Allreduce() 16KB

4.2.2　MPI_Alltoall()

MPI_Alltoall() 関数を用いて MPI_Allreduce() と同様な実験を行った．対応するアルゴリズムの名称を表 3 に示す．AL5 はプロセス数が偶数のときのみ動作し，プロセス数が奇数のときは動作しない．

AL0	default
AL1	Basic Linear
AL2	Pairwise
AL3	Modified Bruck
AL4	Linear with Sync
AL5	Two Proc Only

表 3　MPI_Alltoall() アルゴリズム対応表

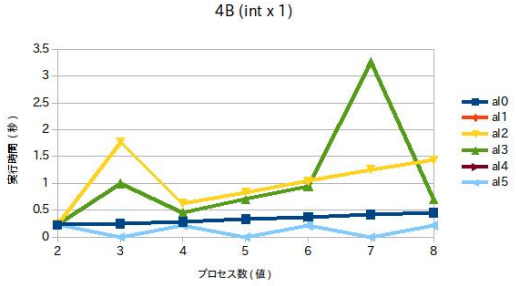

図 3　MPI_Alltoall() 4Bytes

データサイズが 4Bytes の場合の実験結果を図 3 に示す．AL2，3 はこのデータサイズには適していない．AL0，1，4 は同じような挙動を示す．プロセス数が偶数のときは AL5 が最も速いことがわかる．

4.3　最短経路問題

表 4 の瀬戸市道路網，カリフォルニア州道路網，全米道路網を対象とし実験した．通信アルゴリズムと計算機台数を変化させた．

4.3.1　MPI_Allreduce()

MPI_Allreduce() のアルゴリズムを変更しながらカリフォルニア州道路網を対象とした場合の実験結果を図 4 に示す．プロセス数が 6，7 の時，AL5 が最も速い．プロセス数が 2^n でない時，実行時間がかかることがわかる．

ファイル名	ノード数	総枝数	ノードに対する枝数
瀬戸市	58168	62102	1.607631688
カリフォルニア州	1890815	4657742	2.463351518
全米	23947347	58333344	2.43590006

表 4　最短経路問題実験に用いた道路網

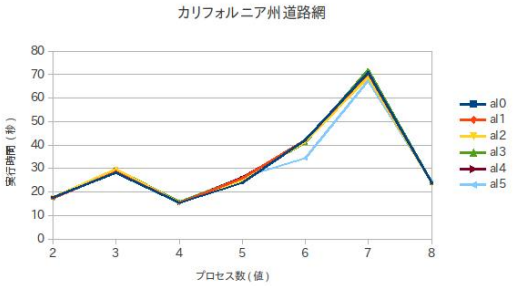

図 4　MPI_Allreduce() カリフォルニア州道路網

4.3.2　MPI_Alltoall()

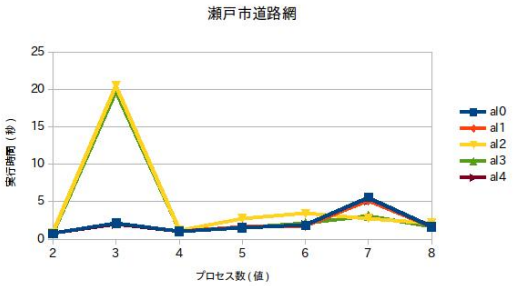

図 5　MPI_Alltoall() 瀬戸市道路網

MPI_Alltoall() のアルゴリズムを変更しながら瀬戸市道路網を対象とし計測した場合の実験結果を図 5 に示す．プロセス数が 7 の時は AL2，3 が速いことがわかる．

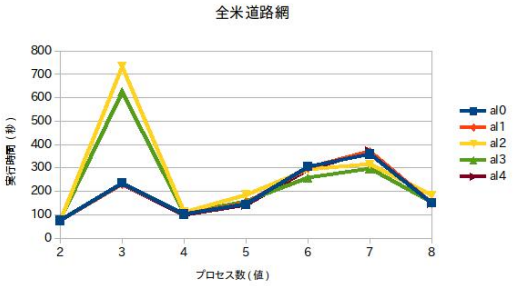

図 6　MPI_Alltoall() 全米道路網

図 6 は全米道路網を対象とした場合である．プロセス数が 6，7 の時は AL3 が最も速いが，依然としてプロセス数が 3 のとき非常に時間がかかることがわかる．

5　考察

実験して得た結果をもとにした考察について述べる．

5.1　デフォルトのアルゴリズム

実験結果から MPI_Allreduce() での default のアルゴリズムはデータサイズによって切り替えられていることが分かる．通信するデータサイズが 4Bytes〜4KB 間の default は AL3 の Recursive Doubling が用いられている．しかし，必ずしも default が最適であるとは言えず他のアルゴリズムを使用したほうが速い場合がある．

5.2　最適なアルゴリズム

MPI_Allreduce() についてデータサイズごとの最適なアルゴリズムを表 5 にまとめた．

データサイズ	アルゴリズム
4Bytes.	Basic Linear, Reduce + Bcast, Recursive Doubling
16B〜64B	Ring, Segmented Ring
256B〜4KB	Basic Linear, Recursive Doubling
16KB	Recursive Doubling
64KB〜1MB	Ring, Segmented Ring

表 5　データサイズ別の最適なアルゴリズム

データサイズが 4Bytes でプロセス数が 3 の時は Basic Linear または Reduce + Bcast が有効である．データサイズが 256Bytes〜4KB でプロセス数が 2 の時は Recursive Doubling が適している．4Bytes〜4KB 間での default アルゴリズムは Recursive Doubling でなので適宜アルゴリズムを表 5 のように変更する必要がある．64KB でのアルゴリズムも Recursive Doubling に変更すべきである．

一方，MPI_Alltoall() では 4〜16Bytes のプロセス数が偶数の時は Two Proc Only に変更するべきである．

5.3　最短経路問題と Allreduce の関係性

実験結果より瀬戸市道路網，カリフォルニア州道路網では，アルゴリズムの切り替えによって実行時間が短くなっているが，全米道路網ではあまり効果が見られなかった．表 5 と最短経路問題の実験結果から，MPI_Allreduce() は通信するデータ量が多いときに安定して動作するようになっていると推測できる．

5.4　最短経路問題と Alltoall の関係性

4 章の実験で MPI_Alltoall() のアルゴリズムを変更することで，ある特定の条件下でかなりの効果を発揮することが分かった．各道路網の最短経路問題を解く際にプロセス数が 6，7 の時，アルゴリズムを切り替えることで実行時間を削減できる．また，default では条件に応じて最適なアルゴリズムに切り替えているはずだが，必ずしも最適でないことが分かる．

6　まとめと今後の課題

OpenMPI の集団通信についてアルゴリズムや通信データ量，プロセス数による性能の変化を調べた．実験の結果から，集団通信のアルゴリズムは，データのサイズなどによって切り替えられているが必ずしもうまく行われるとは限らないことがわかった．

今後は，MPI_Alltoallv() についても同様の実験を行う．また，通信するデータのサイズ及びプロセス数によって，アルゴリズムの切り替えがうまく行われていない場合があるので，これらを解消するように OpenMPI の改良を検討する．

また，単純なアプリケーションの場合と異なり，複雑なアプリケーションを実行する場合，アプリケーションの通信パターンを考慮に入れて分析を進めていく必要がある．

参考文献

[1] A. Crauser, K. Mehlhorn, U. Meyer, and P. Sanders: *A parallelization of dijkstra's shortest path algorithm*. Lecture Notes in Computer Science, L. Brim, J. Gruska, and J. Zlatuska, Eds., vol. 1450., August 1998, pp. 722-731.

[2] 片桐孝洋：“スパコンプログラミング入門 並列処理と MPI の学習” 三美印刷株式会社，東京，2013.

[3] 松田元彦，石川裕，工藤知宏，児玉祐悦，高野了成：“グリッド上のコレクティブ通信のアルゴリズム” 情報処理学会研究報告，HPC-15，pp257-262，(2006-07).

[4] 森江善之，末安直樹，松本 透，南里豪志，石畑宏明，井上弘士，村上和彰：“通信タイミングを考慮した衝突削減のための MPI ランク配置最適化技術” 情報処理学会論文誌，HPC-109，(2007-08).

[5] 柴村英智：“インターコネクトシミュレータ NSIM” マルチコアクラスタ性能 WG 成果報告書，(2013-05)

[6] The Message Passing Interface(MPI) Forum Home Page: `http://www.mpi-forum.org`.

[7] William Gropp,Ewing Lusk and Skjellum: *Using MPI*, The MIT Press,1999

[8] William Gropp,Ewing Lusk and Thakur: *Using MPI-2*, The MIT Press,1999