Vol.2015-ARC-216 No.11
2015/8/4

RDMA における同期通信のインターコネクトシミュレーション

薄田竜太郎[†1]　森江善之[†1]　南里豪志[†1]　柴村英智[†2]

アプリケーションにおいて RDMA を用いる場合，他のプロセス間によるデータアクセスのタイミングを調整するために同期通信が不可欠である．本講演ではインターコネクトシミュレータ「NSIM-ACE」に実装した同期通信のシミュレーションについて，シミュレーション精度及び代表的な同期アルゴリズムの性能比較を報告する．

Interconnection Network Simulation of Synchronization Communication in RDMA

RYUTARO SUSUKITA[†1]　YOSHIYUKI MORIE[†1]　TAKESHI NANRI[†1]
HIDETOMO SHIBAMURA[†2]

If we use RDMA in an application, we inevitably need synchronization communication for controlling the timing of data access from other processes. In this paper, we report simulation accuracy on synchronization communication implemented in NSIM-ACE interconnect simulator. Also we compare performance of commonly-used synchronization algorithms of the implementation.

1. はじめに

Remote Direct Memory Access (RDMA)は並列コンピュータシステムを構成するノード間の通信において，送信ノードのメモリから受信ノードのメモリへプロセッサを介さずに直接データを転送する機能である．最近の並列コンピュータシステムにおいてノード間を接続するインターコネクトに使用されている Network Interface Card (NIC)は RDMA 機能を備えている．RDMA には

(1)　送信ノードのメモリから受信ノードのメモリへ直接データを転送するため，通信レイテンシを短縮できる．

(2)　プロセッサを介さないため，効率よく並列アプリケーションにおける通信以外の演算処理と並列に通信できる．複数の NIC が通信を並列に行う場合もプロセッサが逐次的に処理することがない．

(3)　通信の途中でバッファを必要としないために最小限のメモリ使用量ですむ．

などの利点がある．現在の並列プログラミングで事実上の標準となっている Message Passing Interface (MPI)がメッセージパッシングモデルを基盤としているため，RDMA を基盤とするプログラミングモデルは主流ではないものの，上記の RDMA の利点により，今後 RDMA の重要性が増す可能性がある．RDMA を基盤とするプログラミングモデルを採用している並列化通信ライブラリとしては ARMCI[1], GASNet[2]などがあげられる．これらの通信ライブラリは RDMA を基盤とする put/get 型のインターフェースを備えている．MPI もまたメッセージパッシング以外に put/get 型の通信が可能である．最近の例では ACP ライブラリの基本層が RDMA を基盤とするプログラミングモデルをサポートしている[3][4][5].

反面，RDMA を基盤とするプログラミングモデルでは通信前に同期をとる必要がある場合がある．例えば put 型の通信では受信プロセスが受信操作をすることなく，送信プロセスの送信操作のみで受信ノードのメモリへデータを転送することができるため，受信プロセスが受信準備を完了するまで，送信プロセスは送信を待たなければならない場合が多い．その場合，送信プロセスと受信プロセスは通信前に同期をとる必要がある．また get 型の通信では送信プロセスが送信操作することなく，受信プロセスの受信操作のみで送信ノードのメモリからデータを転送することができるため，送信プロセスがデータを準備するまで待たなければならないことが多い．その場合もやはり送信プロセスと受信プロセスは通信前に同期をとる必要がある．送信プロセスや受信プロセスが複数ある場合は 3 つ以上のプロセス間で同期をとる必要がある．多くのプロセスを同期する場合は，同期に参加する他のプロセスがすべて同期点に達するまで待つバリア同期が一般に使われる．以下，このバリア同期を同期と呼ぶ．

同期による待ち時間や同期を行うための通信自体に必要な時間は RDMA を利用したアプリケーションの性能に影響するため，ノード間通信を行う通信ライブラリの設計や効率よく動作するアプリケーションの開発には同期通信や同期通信を含む通信パターンの性能予測や実機における内部動作の解析が重要である．同期通信自体はノード間の最小レイテンシやバンド幅をパラメータとする数式モデルを構築して性能予測や内部動作の解析を行うことも可能で

†1 九州大学
Kyushu University
†2 (公財)九州先端科学技術研究所
Institute of Systems, Information Technologies and Nanotechnologies (ISIT)

ある．しかし，ノード間で同時に大量のデータを通信し，インターコネクト上で通信衝突が発生するような通信と同期通信の両方を含む通信パターンでは通信衝突によって通信性能が理論値よりも低下するおそれがあり，かつ同期通信による待ち時間も発生するため，このような通信パターンでは通信衝突による性能低下と同期待ち時間を考慮したインターコネクトシミュレーションが性能予測や内部動作の解析に役立つ．

ユーザに使いやすい形で RDMA による通信をシミュレートできるインターコネクトシミュレータとして我々が開発した NSIM-ACE[6]がある．他にも多くのインターコネクトシミュレータが開発されている[7][8][9][10][11]が，これらはユーザに使いやすい形で RDMA を直接シミュレートする機能をもっていない．本論文では NSIM-ACE を使った RDMA の同期通信のインターコネクトシミュレーションについて論述する．

2. NSIM-ACE の概略

NSIM-ACE は MPI と互換性をもったプログラムを入力として受け付け，ユーザが使いやすい高性能な大規模インターコネクトシミュレータ NSIM[12]を拡張して，RDMA のシミュレーションも可能にしたシミュレータである．

2.1 NSIM-ACE のネットワークモデル

NSIM-ACE はシミュレーション対象のネットワークでシステムノードが接続され，接続される各ノードはプロセッサと NIC から構成されているとする．ネットワークはルータ及び，ルータ―ルータ間または NIC―ルータ間のリンクから構成されるとモデル化される．ルータモデルとして，静的ルーティング，バーチャルカットスルー，パイプラインルータを仮定している．NSIM-ACE はファットツリーネットワークを含む多様なネットワークトポロジーをサポートしている．リンク上のデータ転送は基本的にパケット単位でシミュレートされるが，フリットレベルでシミュレーションした場合と同等の精度を持つよう工夫している．メモリ使用量を抑制するため，データ転送においてはデータ量の情報のみ転送され，実際のデータの内容は転送されない．

2.2 NSIM-ACE の RDMA モデル

NSIM-ACE では本論文で利用している put 型の RDMA について以下のようにモデル化している．

put は送信プロセスが，送信ノードのメモリ上におかれたデータを受信ノードのメモリ上へ転送する RDMA である．送信ノードのメモリ上におかれたデータは DMA により送信 NIC へ転送される．送信 NIC はデータをパケットに分解し，ネットワークへ送出する．パケットはネットワーク上を転送されて受信 NIC へ到達する．ネットワーク上のパケット転送については NSIM と同じようにモデル化されている．受信 NIC はパケットからデータを再構成して，DMA により受信ノードのメモリ上へ転送する．受信 NIC はその後コントロールパケットを送信ノードへ向け返送する．コントロールパケットが送信ノードの NIC へ到達すると put が完了する．

2.3 NSIM-ACE の入出力

NSIM-ACE における MPI と互換性をもった入力は MPI プログラムにおいて MPI 関数の接頭辞 MPI_を MGEN_に置き換えたものである．RDMA による並列プログラムは RDMA を行う ACP ライブラリ基本層の関数と類似の機能を持つ関数で記述されたプログラムである．これを MGEN プログラムと呼ぶ．MPI 関数や RDMA 以外の計算部分に関しては実際に計算を行うコードの代わりに MGEN_Comp(t) (t は計算時間の予測値)という形で表現される．これにより，プロセス間の通信開始時刻のずれや計算時間を考慮したシミュレーションを行うことができる．NSIM-ACE はこの MGEN プログラムに従って，インターコネクトをシミュレートし，シミュレーション結果として，予測される全実行時間やその他詳細な統計データを出力する．

3. NSIM-ACE における put 記述の改良

前報告[6]では NSIM-ACE は MGEN プログラムで put を記述する場合は受信プロセスで明示的に受信操作とコントロールパケットの送出を記述していた．しかし，put データの受信よりも受信操作の方が遅ければ，コントロールパケットの送出が受信操作まで遅れることになり，前節で述べた put のシミュレーションモデルとの差異が生じる．そこで受信プロセスにおける記述がなくても put データを受信した後にコントロールパケットを送出するよう NSIM-ACE の動作を変更し，MGEN プログラムで put とポーリングを行う 2 つの関数を NSIM-ACE に追加した．

(1) MGEN_acp_handle MGEN_rdma_put(int dest_rank, int data_size, int tag);

MGEN_rdma_put はランク dest_rank のプロセスへ data_size バイトのデータを put により転送する．受信プロセスで put を区別するためにタグ tag を指定する．本関数は put が完了しなくても終了し，起動した put に対応するハンドルを返す．

(2) void MGEN_rdma_poll(int tag);

MGEN_rdma_poll は本関数を発行したプロセスを受信プロセスとし，タグ tag をもつ put のデータが受信ノードのメモリに転送されるまで待つ関数である．

その結果，シミュレーションモデルとの差異が生ずる場合がなくなり，put データの受信まで待つ場合は受信操作の代わりに MGEN_rdma_poll を発行することで記述できるようになった．

4. 同期通信のシミュレーション

本節では RDMA における同期通信のアルゴリズム及び同期通信の NSIM-ACE への入力について説明する．

4.1 put による同期通信の記述

本論文で扱う同期通信は put により実装されている．前節で述べた put 記述の改良により，put による同期通信のシミュレーションにおいてモデル通りの動作が記述可能となった．また複数の put データの到着を選択して待つ必要があるアルゴリズムについても put データのタグを指定することで記述可能になっている．

4.2 ring アルゴリズム

p 個のプロセスを同期するアルゴリズムはいくつかのものが知られている．同期時間が $O(p)$のアルゴリズムとして ring がある．RDMA の場合，ring アルゴリズムは以下のように実現できる．各プロセスは自分よりランクが 1 大きいプロセスに(ランク $p-1$ のプロセスはランク 0 のプロセスに)何らかのデータを put する．そして自分よりランクが 1 小さいプロセスから(ランク 0 のプロセスはランク $p-1$ のプロセスから)データを受信するまでポーリングして待つ．このステップを $p-1$ 回繰り返すと同期が完了する．

ring アルゴリズムで同期を行う MGEN プログラムを図 1 に示す． MGEN_acp_complete は指定したハンドルに対応する put が完了するまで，つまり put を発行したノードの NIC にコントロールパケットが到達するまで待つ関数である．この MGEN プログラムでは最終ステップの put が完了するのを待つ．

```
#include “mgen.h”
int MGEN_Main(int argc, char **argv){
   int rank, size, ms=8, tag = 0, i;
   MGEN_Comm com = MGEN_COMM_WORLD;
   MGEN_acp_handle_t handle;
   MGEN_Comm_rank(com, &rank);
   MGEN_Comm_size(com, &rank);

   for(i = 0; i < size – 1; i ++){
      handle = MGEN_rdma_put((rank + 1) % size, ms,
tag);
      MGEN_rdma_poll(tag);
   }
   MGEN_acp_complete(handle);
   return(0)
}
```

図 1　ring アルゴリズムの MGEN プログラム

4.3 recursive doubling アルゴリズム

同期時間が $O(\log p)$のアルゴリズムとして recursive doubling がある．RDMA の場合，recursive doubling アルゴリズムは以下のように実現できる．プロセス数が 2^n の場合，recursive doubling アルゴリズムは n ステップで同期を行う．i 番目($i = 1, 2, 3, \ldots, n$)のステップではプロセスをランク順に 2^i 個ずつのグループに分ける．各グループにおいてランクが小さい方の 2^{i-1} 個のプロセスはそれぞれ自分よりランクが 2^{i-1} 大きいプロセスへ何らかのデータを put する．ランクが大きい方の 2^{i-1} 個のプロセスはそれぞれ自分よりランクが 2^{i-1} 小さいプロセスにデータを put する．そして互いにデータを受信するまで待つ．n 番目のステップが完了すると p 個のプロセスが同期される．

プロセス数が 2 のべき乗でない recursive doubling アルゴリズムは，$p = 2^n + r$ とすると，まずランクが 2^n 以上のプロセスがそれぞれ自分よりランクが 2^n 小さいプロセスへデータを put する．ランクが r 未満のプロセスはそれぞれ自分よりランクが 2^n 大きいプロセスからデータを受信するまで待つ．次に，ランクが 2^n 未満のプロセスがプロセス数が 2 のべき乗の場合の recursive doubling アルゴリズムで同期を行う．最後にランクが r 未満のプロセスがそれぞれ自分よりランクが 2^n 大きいプロセスへデータを put する．ランクが 2^n 以上のプロセスはそれぞれ自分よりランクが 2^n 小さいプロセスからデータを受信するまで待つ．これにより p 個のプロセスが同期される．従って，2 のべき乗でない場合の recursive doubling アルゴリズムは 2 のべき乗の場合よりもステップ数を要する．

プロセス数が 2 のべき乗の recursive doubling アルゴリズムで同期を行う MGEN プログラムを図 2 に示す．この MGEN プログラムでは最後のループで各ステップで発行した MGEN_rdma_put に対応するハンドルを指定して MGEN_acp_complete を呼ぶことにより，すべてのステップの put が完了するのを待つ．

```
#include “mgen.h”
int MGEN_Main(int argc, char **argv){
   int rank, size, mask, mod, src, dst, ms=8, tag, i = 0;
   MGEN_Comm com = MGEN_COMM_WORLD;
   MGEN_acp_handle_t handle[100];
   MGEN_Comm_rank(com, &rank);
   MGEN_Comm_size(com, &rank);

   mask = 0x1;
   tag = rank;
   while (mask <= size / 2) {
      mod = rank % (mask * 2);
      if (mod < mask)
         dst = src = rank + mask;
      else
         dst = src = rank – mask;
      handle[i ++] = MGEN_rdma_put(dst, ms, tag);
      MGEN_rdma_poll(src);
      mask <<= 1;
   }

   mask = 0x1;
   i = 0;
   while (mask <= size / 2) {
      MGEN_acp_complete(handle[i ++]);
      mask <<= 1;
   }
   return(0);
}
```

図 2　recursive doubling アルゴリズムの MGEN プログラム

5. シミュレーション精度に関する評価実験

本節では同期通信のシミュレーション精度を評価するためにシミュレーションと実機上の測定結果を比較した実験について述べる．

5.1 実験方法

本実験では，ring と recursive doubling アルゴリズムの同期時間について，シミュレーション結果を実機における測定結果と比較した．両アルゴリズムのプログラムは ACP ライブラリ基本層を用いて書かれたものを利用した．本実験で利用したのは InfiniBand 上の実装である．InfiniBand 上の実装では起動時にプロセスあたり通信スレッドが 1 つ生成され，実行中状態を監視しながら動作しているため，コア数の 1/2 を超えるプロセス数で同期を実行すると通信スレッドの動作が同期時間に影響を及ぼす．その影響を排除するため，コア数の 1/2 のプロセス数で同期を実行した．本実験では平均同期時間の測定値に通常以上のばらつきが見られたため，同期を 100 回実行し，同期に最小限必要と思われる最小同期時間を測定した．

実機には富士通 PRIMERGY RX200 S7 を用いた．ノード数は 16 で，各ノードに 4 コア Intel Xeon プロセッサ E5-2609 (2.40 GHz) が搭載されている．ノード間は InfiniBand QDR スイッチで接続されている．スイッチのスループットは片方向 4.0 GB/s，スイッチ内部におけるポート間レイテンシは 140 ナノ秒以下，ルーティングは destination based routing である．

5.2 シミュレーションパラメータの設定

シミュレーションの設定パラメータを表 1 に記す．ノードの DMA 転送速度は実機において 1 ノード内で 2 プロセスがノード内通信を実行した場合のバンド幅を測定するベンチマークを実行し，その測定値から求めた[6]．メモリバンド幅はネットワークのバンド幅よりもはるかに大きく，シミュレーション結果にほとんど影響しないため，∞に設定した．また通信ライブラリのオーバーヘッドは，0 に設定して無視した場合のシミュレーションとフリーパラメータとして 0.8 マイクロ秒から 1.2 マイクロ秒の値を設定したシミュレーションを行った．本実験環境では 1 マイクロ秒前後の短時間のオーバーヘッドを精度よく測定することが困難であったため，上記のように設定したが，通信ライブラリのオーバーヘッドは実機における測定値に基づいて設定することも可能である．NSIM-ACE の設定パラメータにおける通信ライブラリのオーバーヘッドは通信ライブラリによって生じるオーバーヘッド以外にメモリアクセスの最小レイテンシやノードの DMA 転送の最小レイテンシなどノード内のテータ転送における最小レイテンシを含むため，以下ノードレイテンシと呼ぶ．DMA 転送速度と通信ライブラリのオーバーヘッド以外のパラメータは実機の仕様に基づいて値を決定した．

表 1　NSIM-ACE の設定パラメータ

種別	パラメータ	値
ルータ	ネットワークの最大理論通信速度	4.0 GB/s
	スイッチスループット	4.0 GB/s
	ルーティング計算時間	4.0 ns
	仮想チャネル割り当て時間	4.0 ns
	スイッチ割り当て時間	4.0 ns
	スイッチレイテンシ	128 ns
	ケーブルレイテンシ	0.6 ns
ノード	DMA 転送速度	2.8 GB/s
	メモリバンド幅	∞
	通信ライブラリのオーバーヘッド	本文参照
	プロセス数	1 プロセス/ノード

5.3 実機とシミュレーションの相違点

シミュレーションと実機上の測定にはいくつか異なる点が存在する．1 つはプログラムである．MGEN プログラ

ムは実機のプログラムから通信に関する部分を抽出して作成したが，NSIM-ACE はノードあたり 1 プロセスを仮定しているため，ノードあたり 2 プロセスを実行するかわりに 1 つのプロセスが 2 プロセス分の通信を並列に実行するように記述した．また，本シミュレータはノード内通信をシミュレートしないため，ring アルゴリズムの場合は，同一ノード上の 2 プロセスのうち，ランクが小さい方のプロセスからランクが 1 大きいプロセスへ put する通信がノード内通信となるためシミュレートされない．recursive doubling アルゴリズムの場合はプロセス数が 2 のべき乗の場合における最初のステップがランクが 1 異なるプロセス間の通信でノード内通信となるためにシミュレートされず，同期時間をやや短く予測している．

5.4 実験結果と考察

図 3，図 4 にシミュレーションと実機を比較した結果を示す．ring，recursive doubling いずれのアルゴリズムも実機に比較してノードレイテンシを無視したシミュレーションの同期時間が短いことがわかる．ノードレイテンシを設定した場合は ring，recursive doubling のいずれの場合も 1.0 マイクロ秒に設定したシミュレーションが最も実機の傾向に近い．従って，実機におけるノードレイテンシは 1.0 マイクロ秒前後と推定される．実機において 2 つのアルゴリズムの同期時間を比較すると，予想通り recursive doubling の方が短く，シミュレーション結果も recursive doubling の方が短い．また実機では recursive doubling においてプロセス数が 2 のべき乗の場合がそうでない場合よりも同期時間が短いが，シミュレーションも同じ傾向を示す．これは 2 のべき乗の場合のほうが同期に必要なステップ数が少ないためである．

本実験結果から，ノードレイテンシに適切な値を設定すれば，NSIM-ACE のシミュレーション精度は同期通信のアルゴリズム性能を定量的に比較するために十分であることがわかった．さらに recursive doubling アルゴリズムの実験結果ではプロセス数による同期時間の増減までかなりの程度アルゴリズムの特性を再現することに成功している．同期通信は通信衝突がほとんどないため数式モデルによる性能予測も可能であるものの，NSIM-ACE によるシミュレーションが同期通信の性能予測に有効であると考えられる．

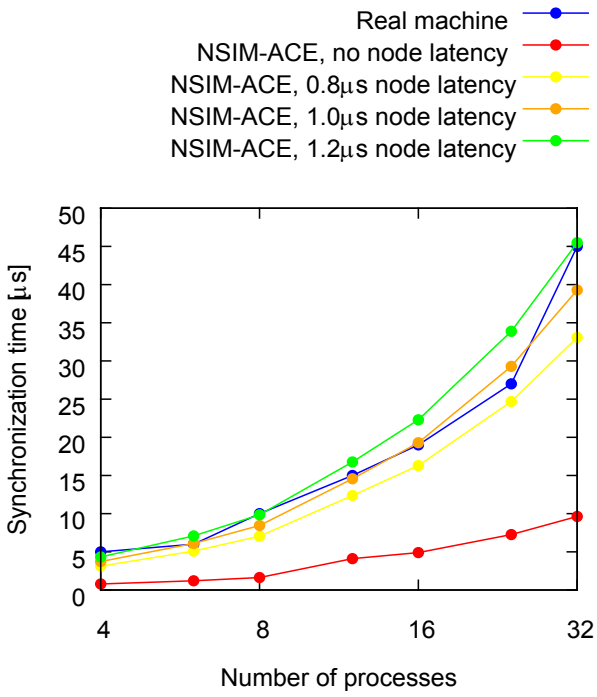

図 3　ring アルゴリズムによる同期時間

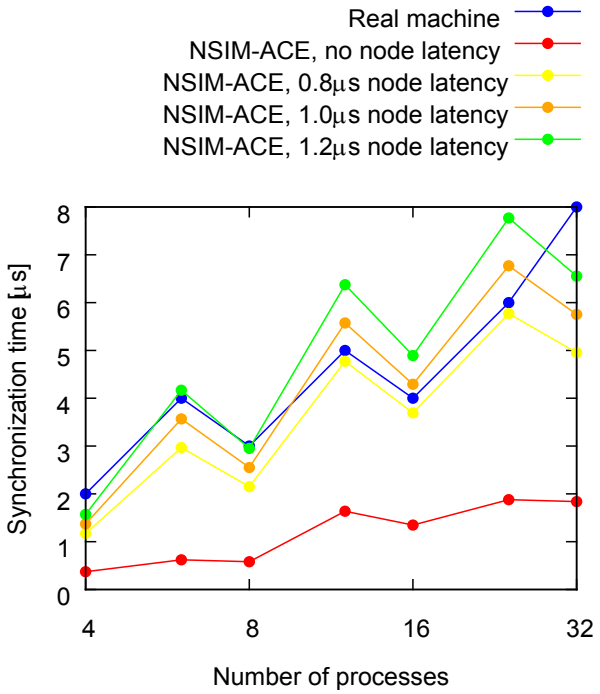

図 4　recursive doubling アルゴリズムによる同期時間

6. 実アプリケーションにおける RDMA の同期通信

本節では実アプリケーションにおいて同期通信を含む RDMA 通信のシミュレーション精度を評価するためにシミュレーションと実機上の測定結果を比較した実験につい

て述べる．

6.1 実験方法

本実験では，重力 N 体シミュレーションに現れる領域分割後の粒子データの通信時間について，シミュレーション結果と実機における測定結果と比較した．重力 N 体シミュレーションではシミュレーション領域を計算時間が均等になるよう分割し，各プロセスは分割された領域に含まれる粒子の計算を担当する．シミュレーション中に他のプロセスの担当領域へ粒子が移動して，計算時間の不均等が大きくなると，計算時間が均等になるように領域分割を再び行う．領域分割後に担当プロセスが変更になった粒子があれば，新しい担当プロセスへ粒子データの通信を行う．粒子データの送信側は各粒子の位置データと領域分割情報からその粒子をどのプロセスへ送信すべきか決定できるが，受信側はいくつのプロセスから粒子データを受信するか，どのプロセスから粒子データを受信するかを予測することが難しい．このような通信パターンは送信プロセスが粒子データを put し，put が完了した後で全プロセスの同期をとることで実現できる．

領域分割後の粒子データの通信量はプロセスあたり数 MB に達する場合があり，このような通信を多数のプロセスが同時に行うと通信衝突が発生して，数式モデルによる性能予測では不十分であるおそれがある．

実機の領域分割プログラムは ACP ライブラリ基本層を用いて実装した．プログラム中の同期は recursive doubling アルゴリズムで書かれたものを利用した．重力 N 体シミュレーションコード GADGET-2[13]で 128^3 粒子のシミュレーションを行い，領域分割前後の 2 つのスナップショットを出力したものを粒子データとして用いた．領域分割プログラムにおける粒子データは粒子あたり 48 バイトである．本実験でも領域分割後の粒子データの平均通信時間の測定値に通常以上のばらつきが見られたため，同じ領域分割を 20 回実行し，粒子データの通信に最小限必要と思われる最小実行時間を測定した．実機には前節と同じ富士通 PRIMERGY RX200 S7 を用い，ノードあたり 2 プロセス，全 32 プロセスで実行した．シミュレーションの設定パラメータは表 1 と同じ値を設定し，ノードレイテンシは 1.0 マイクロ秒に設定した．

6.2 通信以外の処理時間

粒子データを通信するためにはまず他のプロセスへ送信する粒子数を求める．次に各プロセスで粒子データの通信準備が完了するのを待つために全プロセス間で同期を行う．そして各プロセスは同じプロセスへ送信する粒子データが送信バッファ上で連続するようにコピーするパックを行った後，粒子データを新しい担当プロセスへ put する．すべての put が完了した後で全プロセス間の同期を行い，最後に受信バッファから粒子データをコピーするアンパックを行う．MGEN プログラムでは通信以外の 3 つの処理時間は実機の測定値を，送信粒子数を求める処理は各プロセスの担当粒子数及び送信粒子数，パックは送信粒子数，アンパックは受信粒子数を用いて原点を通る 1 次式でフィッティングしたものを用いた．MGEN プログラムで各プロセスの担当粒子数，送信粒子数，受信粒子数を読み込み，フィッティングで得られた式から求めた処理時間を MGEN_Comp で記述した．

実機における測定値とフィッティング結果を図 5 から図 7 に示す．送信粒子数を求める処理及びパックはかなり良好なフィッティング結果が得られている．アンパックは明らかにフィッティングできない測定値が存在したため，これらの測定値を除いてフィティングを行った．

測定値をフィッティングから外したプロセスにおけるアンパック処理を解析したところ，フィッティング線上に近いプロセスに比べて動的に確保したメモリへのアクセスに時間がかかっていることが分かった．プロセスが以前に動的に確保して解放した領域を再利用できず，プロセスが新たに確保した領域にアクセスしているために他のアンパックよりも時間を要している可能性が考えられる．

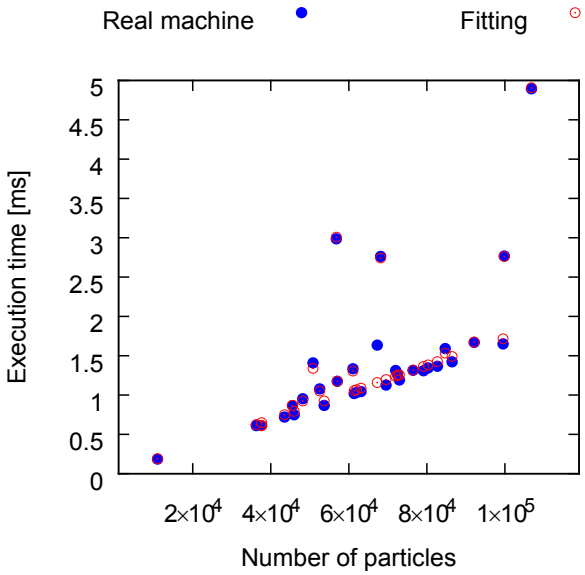

図 5　送信粒子数を求める処理の実行時間

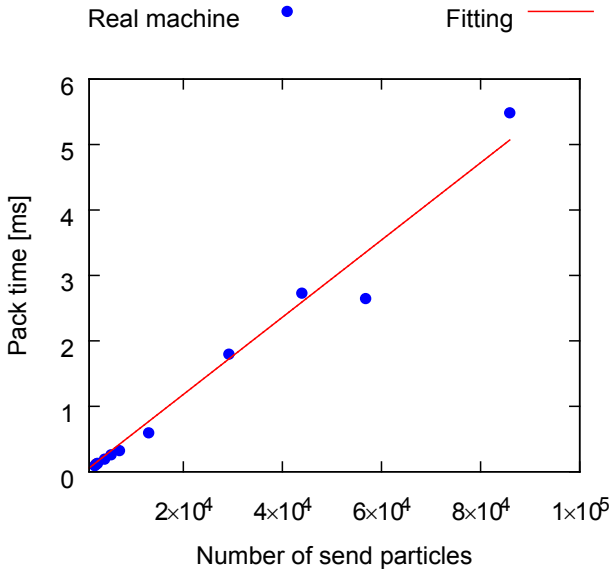

図 6　送信粒子データのパック時間

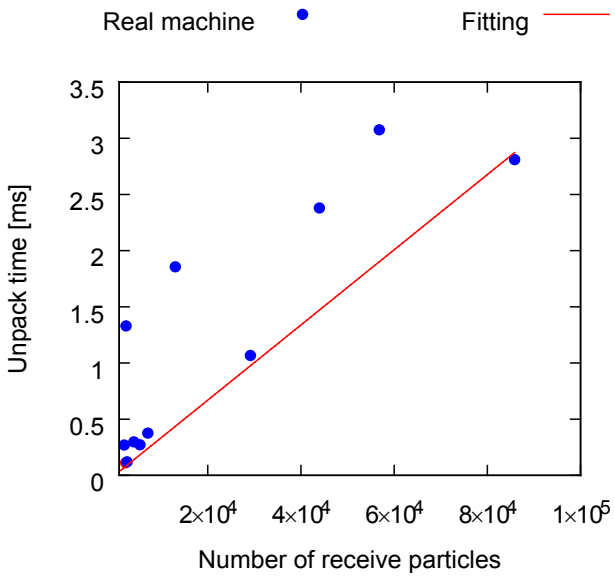

図 7　受信粒子データのアンパック時間

6.3 実機とシミュレーションの相違点

MGEN プログラムは実機の領域分割プログラムから領域分割後のデータ粒子の通信を行う put と同期通信の部分を抽出して作成し，MGEN_comp 関数で通信以外の 3 つの処理時間を記述した．実機のプログラムではこれ以外に ACP ライブラリ基本層のアトミック通信を行っているが，これらの所要時間は put の全通信時間に比べて非常に短いと予想される．また前節と同様にノードあたり 2 プロセスを実行するかわりに 1 つのプロセスが 2 プロセス分の通信を並列に実行するように記述した．この他，同じノード内のプロセスへの put がノード内通信となってシミュレートされないため，実行時間をやや短く予測している可能性がある．

6.4 実験結果と考察

実験結果は実機における実行時間 17.5 ミリ秒に対して，シミュレーション結果は 15.2 ミリ秒であった．誤差は 13%である．また MGEN プログラムにおいて，同期通信をすべて取り去るとシミュレーション結果は 13.2 ミリ秒に減少した．これは送信粒子数を求める処理に要する時間がプロセス間で異なるために発生する同期待ち時間がなくなり，パック時間及び put の通信時間がプロセス間で異なるために発生する同期待ち時間もなくなるためである．

本実験のシミュレーション精度は実アプリケーションに現れる RDMA 通信の性能予測に NSIM-ACE が利用できる可能性を示している．ただし，通信以外の処理に要する時間が put の通信時間よりも長いために，通信以外の処理に要する時間における誤差の方が put よりも全実行時間における誤差への寄与が大きく，put の通信時間における誤差はさらに大きいこともありうる．一方同期を無視したシミュレーションでは誤差が相当増えており，本実験の通信パターンでは同期通信のシミュレーションによって精度が向上していることがわかる．

7. まとめ

本論文ではインターコネクトシミュレータ NSIM-ACE を用いた RDMA による同期通信のシミュレーションについて述べた．シミュレーションモデルとの差異が生じないように NSIM-ACE の改良を行ったため，シミュレーション結果の信頼性が増している．同期通信に関する実験結果は RDMA による同期アルゴリズムの性能比較やアルゴリズム特性を予測する上で NSIM-ACE が有効であることを示し，さらに実アプリケーションに現れる RDMA 通信に関する性能予測にも有効である可能性がある．

謝辞　本研究は，科学技術振興機構 戦略的創造研究推進事業 (CREST)「ポストペタスケール高性能計算に資するシステムソフトウェア技術の創出」研究領域，「省メモリ 技術と動的最適化技術によるスケーラブル通信ライブラリの開発」の一部として実施された．

参考文献

1)　Nieplocha, J. and Carpenter, B.: ARMCI: A Portable Remote Memory Copy Library for Distributed Array Libraries and Compiler Run-time Systems. Proc. RTSPP of IPPS/SDP'99. (1999).
2)　Bonachea, D.: GASNet Specification, v1.1, U.C. Berkeley Tech Report (UCB/CSD-02-1207), (2002).
3)　住元真司, 安島雄一郎, 佐賀一繁, 野瀬貴史, 三浦健一, 南里豪志: エクサスケール通信向け ACP スタックの設計思想, 情報処理学会研究報告, Vol.2014-HPC-143 No.8, (2014).
4)　安島雄一郎, 佐賀一繁, 野瀬貴史, 三浦健一, 住元真司: ACP

基本層の設計思想とインタフェース, 情報処理学会研究報告, Vol.2014-HPC-143 No.9 (2014).
5) 佐賀一繁, 安島雄一郎, 野瀬貴史, 三浦健一, 住元真司: ACP 基本層の実装と初期評価, 情報処理学会研究報告, Vol.2014-HPC-143 No.10 (2014).
6) 薄田竜太郎, 森江善之, 南里豪志, 柴村英智: RDMA 評価のための大規模インターコネクトシミュレータ「NSIM-ACE」, 情報処理学会研究報告, Vol.2014-HPC-147 No.31 (2014).
7) Adiga, N. R. et al.: Blue Gene/L Torus Interconnection Network, IBM J. Res. Dev., Vol.49, pp.265-276, (2005).
8) Choudhury, N., Mehta, Y., Wilmarth, T. L., Bohm, E. J. and Kale, L.V.: Scaling an Optimistic Parallel Simulation of Large-scale Interconnection Networks, Proc. 37th Conference on Winter simulation, Conference, WSC '05, pp.591-600, (2005).
9) Zheng, G., Kakulapati, G. and Kalé, L.V.: BigSim: A Parallel Simulator for Performance Prediction of Extremely Large Parallel Machines, Parallel and Distributed Processing Symposium, International, Vol.1, p.78b, (2004).
10) Kale, L.V. and Krishnan, S.: Charm++: A Portable Concurrent Object Oriented System Based on C++, Proc. Eighth Annual Conference on Object-Oriented Programming Systems, Languages, and Applications, OOPSLA '93, pp.91-108, (1993).
11) Ridruejo, F. J. and Alonso, J. M.: INSEE: An Interconnection Network Simulation and Evaluation Environment, Proc. 11th Euro-Par Parallel Processing Conference 2005, Euro-Par'05, pp.1014-1023, (2005).
12) Miwa, H. et al.: NSIM: An Interconnection Network Simulator for Extreme-Scale Parallel Computers, IEICE Trans. Inf. & Syst., Vol.94, No.12, pp.2298-2308, (2011).
13) V. Springel:. The Cosmological Simulation Code GADGET-2, Monthly Notices of the Royal Astronomical Society, Vol. 364, pp.1105-1134, (2005).