Vol.2010-HPC-125 No.5
2010/6/17

NSIM: 将来の大規模相互結合網を対象とした通信シミュレータの開発

三輪英樹[†] 薄田竜太郎[††] 柴村英智[††] 平尾智也[††]
眞木淳[††] 稲富雄一[†††] 井上弘士[†††]
安島雄一郎[†] 三吉郁夫[†] 清水俊幸[†] 安藤壽茂

近い将来，数 10 万ノードを相互結合網によって接続した大規模並列計算機が主流となると予測される．大規模並列計算機向けに通信ライブラリ，通信アルゴリズム，および相互結合網の開発を効率的に進めるためには，相互結合網の性能予測のための相互結合網シミュレータが必要不可欠である．シミュレータに対する要件としては，相互結合網を適切な詳細度でモデル化し，通信パターンを記述するための柔軟なインタフェースを持ち，シミュレーション結果に関して十分に詳細な情報を出力できることが挙がる．また，近年普及が拡大しているマルチコア・プロセッサの特徴である，共有メモリを介した低いレイテンシでの通信を利用できる設計となっていることが望ましい．そこで，本稿では大規模相互結合網の性能予測のための相互結合網シミュレータ NSIM を提案する．本シミュレータでは，対象システムにおけるプロセス間のデータ転送を省略し，シミュレーション時間とメモリ消費量を削減することで，マルチコア・プロセッサを搭載した小規模な並列計算機上での動作を可能としている．さらに，実行駆動型シミュレーション方式の採用により，相互結合網における混雑状況で振る舞いが動的に変化する通信パターンを正しくシミュレーションすることができる．実機でのランダムリング通信の実行結果との比較により，本シミュレータの予測結果が定性的に正しいことを確認した．また，全対全通信のシミュレーション結果から，本シミュレータが将来の大規模相互結合網を現実的な時間内でシミュレーションする能力を持つこと，および既存シミュレータと比較して省メモリかつ高速に動作することを示した．

NSIM: Communication Simulator for Next Generation Extreme-scale Interconnection Networks

Hideki Miwa[†] Ryutaro Susukita[††] Hidetomo Shibamura[††]
Tomoya Hirao[††] Jun Maki[††] Yuichi Inadomi[†††] Koji Inoue[†††]
Yuichiro Ajima[†] Ikuo Miyoshi[†] Toshiyuki Shimizu[†] and
Hisashige Ando

The future generation network architectures tend to interconnect more than hundreds of thousands of nodes. To efficiently evaluate the performance of such interconnection networks, simulation tools capable of modeling the networks with sufficient details, supporting a user-friendly interface to describe communication patterns, providing the users with enough performance information, running at great speed even on a small-scale parallel machine, are a real necessity. In this paper, we introduce a new interconnection network simulator NSIM, for the evaluation of the performance of extreme-scale interconnection networks. Helped by the execution-driven simulation approach, NSIM can simulate communications whose behaviors are adaptively changed by the network congestion. It also supports a flexible interface to describe the traffic patterns as inputs and can run on workstations with multi-core processors. The experimental results confirmed that this interconnection simulator is capable of ultra large-scale interconnection simulation, consumes smaller memory, and runs faster than the existing simulators.

1. はじめに

大規模並列計算機ではより多くのノードを相互結合網により接続することで総演算性能を高めており，近い将来，数 10 万ノードを接続する大規模並列計算機が主流となると予測される．このような大規模並列計算機向けにアプリケーションの性能チューニングや，通信ライブラリ，通信アルゴリズム，および相互結合網の制御方式の開発を効率的に進めるためには，相互結合網における通信の性能を予測する必要がある．しかしながら，相互結合網の性能は，輻輳やプロセスのロードインバランスなどに影響されるため，静的および定性的に予測することは困難である．したがって，通信関数やスイッチの振る舞い，およびノード間でのパケット転送を的確にシミュレーションできるツール(シミュレータ)が必要となる．その他，通信パターンの記述インタフェースの柔軟性，対応する通信パターンおよび取得可能な情報の多様性，結果の精度，シミュレーション速度なども，シミュレータに求められる要件である．

シミュレーション対象の大規模化に伴い，既存のシミュレータによるシミュレーションが現実的な時間内では困難になる場合がある．例えば，全対全通信のアルゴリズムなどのように，通信量が対象システムのノード数(n)に対し $O(n^2)$ もしくは $O(n \cdot \log(n))$ となる場合である．全対全通信は，気象の全球シミュレーション，および分子動力学シミュレーションなどにおける FFT(高速フーリエ変換)で利用される集団通信である．シミュレーション対象の大規模化に伴う処理量の増加を抑えることは難しいため，シ

† 富士通株式会社
Fujitsu Ltd.
†† 財団法人 九州先端科学技術研究所
Institute of Systems, Information Technologies and Nanotechnologies
††† 国立大学法人 九州大学
Kyushu University

ミュレーションを現実的な時間内で完了させるためには単位処理量あたりのシミュレーション時間を低減しなければならない．このためには既存のシミュレーションモデルの見直しが必要である．

一方，シミュレータの実行環境としてマルチコア・プロセッサを搭載した並列計算機が普及している．特徴は，物理メモリを共有するプロセッサ・コア間では共有メモリを介して低いレイテンシで通信可能であること，および旧来の大規模メモリ共有並列計算機と比較して容易かつ安価に入手できることである．ただし，物理メモリを共有するプロセッサ・コア数は限られているため，低いレイテンシでの通信が可能な並列数に制約があり，利用可能メモリ容量も制限される．また，高い並列性や多くのメモリを利用する場合にはネットワークを介した通信が必要となり，通信レイテンシが高まる．既存のシミュレータは，並列性もしくは低レイテンシ通信のいずれか片方の特徴を利用するように設計されている．例えば，米国 Illinois 大学で開発されている BigNetSim [3] は，高い並列性を利用する設計となっている．動作には多くのメモリを必要とするため，実行時の並列数を落として共有メモリを介した通信のみを利用する場合，メモリ不足となる可能性がある．また，IBM の BlueGene/L 相互結合網シミュレータ [1] は，IBM SP2 のような大規模メモリ共有並列計算機を対象として設計されている．このため，共有メモリを介した通信のみでシミュレーションを進めることはできるが，大規模な並列性は利用できない．並列性と低い通信レイテンシとを両方利用するためには，大規模並列計算機だけでなく小規模メモリ共有並列計算機での動作を考慮してシミュレータを設計することが望ましい．

本稿では，相互結合網シミュレータ NSIM を提案する．本シミュレータでは，数 10 万ノードを接続する将来の相互結合網を対象として，様々な通信パターンを現実的な時間内でシミュレーションすることを目的とする．この実現のため，我々はシミュレーション処理量およびメモリ消費量が減るようにシミュレータを設計した．これにより，小規模メモリ共有並列計算機上での大規模シミュレーションも可能となっている．シミュレーション対象通信パターンは，NSIM が提供する MPI 互換インタフェース (MGEN API)を利用しメッセージパッシング方式の並列プログラム(MGEN プログラム)として記述する(C 言語のみ可能，FORTRAN は未対応)．MGEN プログラムは MPI プログラムと同様の要領で記述できる．MGEN API には MPI には定義されていない拡張関数が含まれており，複数の NIC を同時に利用した通信，ゼロ・コピー通信などを指定したシミュレーションが可能である．さらに，NSIM は実行駆動方式のシミュレータとして実装されているため，ネットワークの混雑状況に応じて振舞いが動的に変化する通信パターンも正しくシミュレーションすることができる．

本稿の構成は以下のとおりである．まず，第 2 節で既存の相互結合網シミュレータを紹介する．次に，第 3 節で NSIM の入出力，内部構成，および動作を説明し，問題解決のための設計上の工夫を説明する．続いて，第 4 節で MGEN API および MGEN プログラムのサンプルを紹介する．そして，第 5 節で NSIM の実装方法を説明し，実装面での工夫について述べる．さらに，第 6 節で実験結果を提示し，NSIM が小規模メモリ共有並列計算機上で 10 万ノード規模のシミュレーションを現実的な時間内で完了できることを示す．

2. 関連研究

本節では，代表的な相互結合網シミュレータを挙げ，その特徴を説明する．N. R. Adiga らによる BlueGene/L 相互結合網シミュレータ [1] はトレース駆動型相互結合網シミュレータである．トレースは IBM のトレースキャプチャーユーティリティと呼ばれるツールにより，擬似コードに基づき生成される．本シミュレータは，共有メモリ計算機上での実行を想定して開発され，64GB メモリを搭載した 16 ウェイ IBM POWER3+ SMP ノード上で動作させている．利用可能な並列度がそれほど高くないことから，高い並列度を要求しない保守的離散事象シミュレーション方式で実装されている．実行時には複数のスレッドが動的に生成され，各スレッドは YAWNS (yet another windowing network simulator) と呼ばれるプロトコル [2] を利用して対象システムの時刻を同期している．

L. V. Kale らは BigNetSim [3]を開発している．本シミュレータは，さまざまなネットワークトポロジの詳細モデルを実装しており，相互結合網，トポロジ，規模，およびレイテンシに関する様々なパラメータを設定可能である．本シミュレータは，POSE (general-purpose optimistically-synchronized PDES (parallel discrete event simulation) environment) [5]を利用し，楽観的離散事象シミュレーション方式で実装されている．本シミュレータは，プロセスを仮想化することにより，数 100 ノード規模の高い並列性を利用できるようになっている．本シミュレータは BigSim [4] により生成されたトレースに基づき相互結合網をシミュレーションする．BigSim は大規模並列計算機のシミュレータであり，相互結合網での通信衝突を考慮しない簡易モデルを備えている．BigSim を利用するには，MPI 等で記述された並列プログラムに Charm++ [6]ライブラリをリンクする．Charm++ライブラリは，あらかじめ BigSim 機能を有効化して構築しておく．ユーザが並列プログラムの実行ファイルを実行することにより，トレースファイルが生成される．BigNetSim はこのトレースファイルを読み込み，詳細なネットワークモデルに基づき相互結合網のシミュレーションを行う．

上記シミュレータと NSIM とで最も異なる点は想定する実行環境である．NSIM はマルチコアを搭載したワークステーションのように並列度が低いメモリ共有計算機での動作を基本としつつ，高い並列性を持つ PC クラスタや大規模並列計算機での動作も可能である．このため，16 ウェイ SMP ノード上での動作を想定している BlueGene/L 相互結合網シミュレータと同様に，保守的離散事象シミュレーション方式を採用して

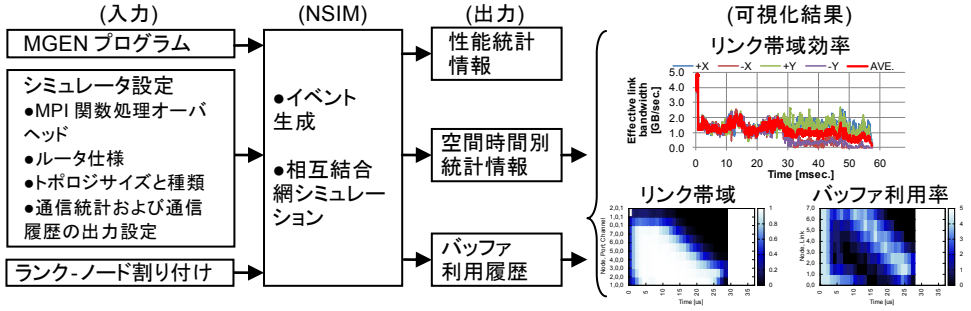

図 2 NSIM の入出力

いる．一方，BigNetSim は比較的高い並列度 (数 100 並列) での実行を想定しているため，並列度を利用しやすい楽観的離散事象シミュレーション方式を採用している．このため，BigNetSim を並列度が低い環境で実行させる場合，物理メモリの不足で実行が継続できない場合や，並列度の不足によりシミュレーションが現実的な時間内に完了しない場合がある．このような状況を避けるために NSIM に導入した設計上および実装上の特徴について以降で説明する．

3. NSIM の設計

本節では，NSIM の内部構成について説明し，第 1 節で挙げた問題点を解決する設計上の手段について述べる．

3.1 対象システムのモデル

NSIM では，k-ary n-cube/mesh トポロジ，および FBB (full bisectional bandwidth) 構成の FatTree トポロジのモデルを実装している．k-ary n-cube/mesh の例としては，3 次元トーラス/メッシュ，および Tofu (6 次元メッシュ) [7]などが該当する．

ノードおよびルータ間は双方向リンクで接続されている．ノードは 1 つのシングルコア CPU を搭載し，主記憶上に 2 つのバッファ(USER バッファおよび MPI バッファ)を搭載している．USER バッファは MGEN プログラム中で確保された変数領域の集合を示す．MPI 関数によるワン・コピー通信時は，当該データが USER バッファから MPI バッファにコピーされた上で転送されることを仮定している．ゼロ・コピー通信では MPI バッファは利用されない．

ルータは，入力バッファ，出力バッファ，およびクロスバスイッチを搭載しており，パイプライン転送 (事前にルーティング計算を行い，先行通信完了直後に転送を開始する) に対応している．ルーティングアルゴリズムとして静的次元オーダルーティング (k-ary n-cube/mesh トポロジの場合) ，上り直進ルーティングおよび下り直進ルーティング (FatTree) を実装している．後者 2 つは FBB 構成の FatTree 用ルーティングアルゴリズムで，送信元ノードからスパインスイッチへ直進する，もしくはスパインスイッチから宛先ノードへ直進するアルゴリズムである．フロー制御方式はバーチャル・カットスルー方式を想定している．

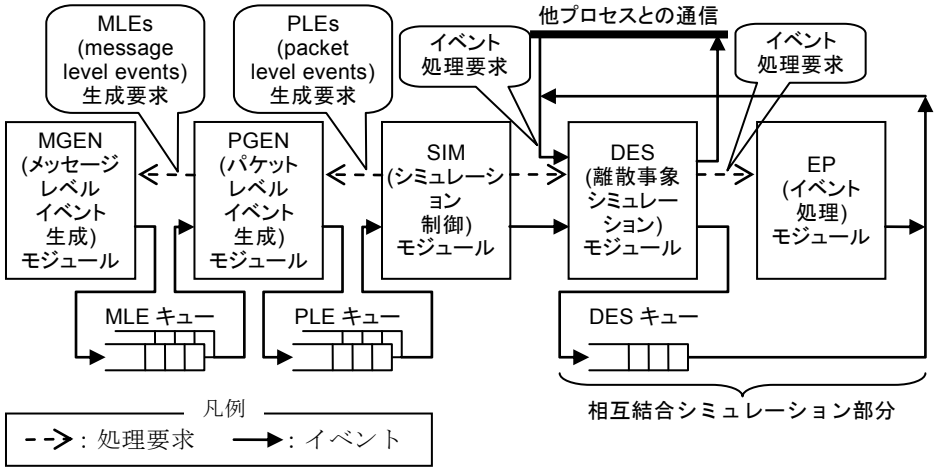

図 1 NSIM の内部処理フロー

データ転送はパケットレベルでモデル化している．フリットレベルのほうが高精度を実現できるが，非常に遅く現実的な時間内でシミュレーションできる規模が限定される．しかしながら，バーチャル・カットスルールーティングについては，パケットレベルでのモデル化でフリットレベルと同等の精度を維持することが可能である．例えば, 現ノードから次ノードへバーチャル･カットスルー方式で転送中のパケットがあると仮定する．先頭フリットが次ノードへ移動し始めると，最終フリットまで後続のフリット転送は中断されない. このため, 次ノードにおける最終フリット到着時刻は，基本的には次ノードにおける先頭フリット到着時刻に後続フリット転送レイテンシを加算することにより計算できる．ただし，NIC とスイッチの転送帯域が異なる場合，現ノードへの最終フリット到着時刻に基づき次ノードでの最終フリット到着時刻を計算し，先に求めた最終フリット到着時刻との最大値を求めなければならない場合がある．いずれにしても，中間フリットの到着時刻の計算を省くことで精度が大きく落ちることはない．したがって，フリット単位でのモデル化ではなく，パケット単位でモデル化した．

3.2 シミュレータの全体構成

図 2 にシミュレータの全体像を示す．図 2 の左側が入力，中ほどがシミュレータ本体，右側が出力を表している．本節では主に入出力について説明し，機能については第 3.3 節にて説明する．

シミュレータへの入力は 3 種類あり，通信パターンを記述した MGEN プログラム，シミュレータ設定ファイル，およびランク - ノード割り付けファイルである．シミュレータはシミュレータ設定ファイルの内容に従って初期化される．シミュレータ設定ファイルでは，MPI 処理時間オーバヘッド，ルータ仕様，ネットワーク仕様などのパラメータを指定することができる．シミュレータ設定ファイルを変更することにより，異なる構成のシステムをシミュレーションすることができる．また，トポロジを意識したアプリケーションや通信アルゴリズムの評価を行う場合には，ランク - ノード割り付けファイルを差し替えることで様々なランクとノードとの割り付けを試すことができる．このファイルは，1 行が 1 レコードとなっており，各レコードにランクとノードアドレスを記述することにより対応関係を記述する．

シミュレータは 3 種類の情報を出力する．性能統計情報，時間空間別統計情報，およびバッファ利用履歴である．これらのうち，後者 2 つはユーザが指定した場合にのみ出力される．まず，性能統計情報は，MGEN プログラム実行時間，平均リンクバンド幅，リンク帯域効率，累積転送データ量などを出力したものである．これらの情報を参照することで，シミュレーション全体の傾向を把握することができる．次に，時間空間別統計情報は，性能統計情報を時間および空間ごとに出力したものである．この情報はスクリプトを利用することで可視化することができる．図 2 の右に，リンク帯域効率，リンクスループット，バッファ利用率を可視化した例を示す．ユーザはこれらの情報に基づき，通信状況を確認および解析することができる．最後に，バッファ利用履歴は，指定ルータの指定期間においてバッファを通過したパケットの情報を出力したものである．この情報を利用することにより混雑の発生原因を解析することが可能である．

3.3 シミュレータ内部構成と処理の流れ

図 1 にシミュレータの内部構成を示す．本シミュレータは，主に 5 つの機能ブロックと 3 種類のキュー(MLE,PLE,DES キュー)から構成されている．図 1 において，機能ブロックは MGEN プログラムに基づきメッセージレベルイベント(MLE)を生成する MGEN モジュール，MLE を分割してパケットレベルイベント(PLE)を生成する PGEN モジュール，シミュレーションを制御する SIM モジュール，離散事象シミュレーション(DES)キューを管理する DES モジュール，およびイベントを処理する EP モジュールである．MLE はメッセージの送受信等に対応するイベントであり，PLE はパケット送受信等に対応するイベントである．以降，イベント処理フローについて説明する．

- **MLE 生成**
  SIM モジュールは PLE キューが空である場合, PGEN モジュールに対して PLE 生成を要求する．PGEN モジュールは MLE キューが空である場合に MGEN モジュールに対して MLE 生成を要求する．MGEN モジュールは MGEN プログラムを呼び出して MLE を生成する．MGEN モジュールは，MLE キューが一杯となるか，もしくは受信関数(ノンブロッキング受信の場合，対応する完了待ち関数)に到達した場合，MLE 生成を停止する．
- **PLE 生成**
  PGEN モジュールは PLE キューが空である場合, MLE キューからイベントを取り出し，PLE キューが一杯になるまで PLE を生成する．PGEN モジュールは各 PLE に対応するパケットがネットワークへ注入される時刻を決定して PLE キューへ詰める．
- **相互結合網シミュレーション**
  SIM モジュールは，PLE キューからイベントを取り出し，イベントがパケット転送に対応する場合 DES モジュールへ渡す．DES モジュールは DES キューの先頭イベントの処理を EP モジュールへ要求する．EP モジュールはイベントに記録されたパケットの情報を読み取り，パケットが宛先ノードに到着していない場合，パケットの次の 1 ホップの転送に必要な資源の空き状況を調べる．資源を確保可能である場合は確保し，転送に必要なレイテンシを計算して転送完了時刻を求める．EP モジュールは，確保した資源を開放するイベントと，次のホップのパケット転送を行うイベントを生成して DES モジュールへ渡す. パケットが宛先ノードに到着している場合，EP モジュールは到着時刻を求め，到着パケットの情報を SIM モジュールへ渡す．パケットがあるメッセージの最終パケットである場合，SIM モジュールは PGEN モジュールを介して MGEN モジュールへメッセージ受信完了を通知する．MGEN モジュールは停止していたイベント生成を再開する．
- **統計情報生成**
  シミュレーションの間，EP モジュールは性能に関する統計情報を収集する．指定された間隔で空間別および時間別の統計値を計算し，空間時間別統計情報としてファイルへ出力する．同時に，指定された間隔でバッファ利用履歴をファイルへ出力する．シミュレーション完了時には，統計値の平均や合計などを求め，性能統計情報としてファイルへ出力する．

本シミュレータは実行駆動方式で実装しており，相互結合網シミュレーションの進捗に合わせて MLE および PLE の生成を行う．このため，マスタプロセスがワーカープロセスに動的にワークロードを割り当てるアプリケーションのように，輻輳および負荷バランス等により振る舞いが変化する通信パターンを正しくシミュレーションすることができる．

(MPI プログラム)

```
#include <stdio.h>
#include "mpi.h"
#define TAG0 0
#define COM MPI_COMM_WORLD
#define MSIZE 1
int main( int argc, char ** argv ) {
    int myrank, mysize, left_rank,right_rank, dest;
    MPI_Status  status;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(COM, &myrank);
    MPI_Comm_size(COM, &mysize);
    int sendbuf[MSIZE], recvbuf[MSIZE];
    left_rank  = (myrank-1+mysize)%mysize;
    right_rank = (myrank+1)%mysize;

    MPI_Send( &sendbuf, MSIZE, MPI_BYTE, left_rank, TAG0, COM );
    MPI_Send( &sendbuf, MSIZE, MPI_BYTE, right_rank, TAG0, COM );

    int i;
    for (i=0;i<mysize*MSIZE;i++) {
        MPI_Recv( &recvbuf, MSIZE, MPI_BYTE, MPI_ANY_SOURCE, TAG0, COM, &status);
        if ( status.MPI_SOURCE == left_rank  ) dest = right_rank ;
        if ( status.MPI_SOURCE == right_rank ) dest = left_rank  ;
        MPI_Send( &sendbuf, MSIZE, MPI_BYTE, dest, TAG0, COM);
    }
    MPI_Recv( &recvbuf, MSIZE, MPI_BYTE, MPI_ANY_SOURCE, TAG0, COM, &status);
    MPI_Recv( &recvbuf, MSIZE, MPI_BYTE, MPI_ANY_SOURCE, TAG0, COM, &status);
    MPI_Finalize();
    return 0;
}
```

(MGEN プログラム)

```
#include <stdio.h>
#include "mgen-mpi.h"
#define TAG0 0
#define COM MGEN_COMM_WORLD
#define MSIZE 1
int main( int argc, char ** argv ) {
    int myrank, mysize, left_rank,right_rank, dest;
    MPI_Status  status;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(COM, &myrank);
    MPI_Comm_size(COM, &mysize);
    int sendbuf[MSIZE], recvbuf[MSIZE];
    left_rank  = (myrank-1+mysize)%mysize;
    right_rank = (myrank+1)%mysize;

    MPI_Send( &sendbuf, MSIZE, MPI_BYTE, left_rank, TAG0, COM );
    MPI_Send( &sendbuf, MSIZE, MPI_BYTE, right_rank, TAG0, COM );

    int i;
    for (i=0;i<mysize*MSIZE;i++) {
        MPI_Recv( &recvbuf, MSIZE, MPI_BYTE, MPI_ANY_SOURCE, TAG0, COM, &status);
        if ( status.MPI_SOURCE == left_rank  ) dest = right_rank ;
        if ( status.MPI_SOURCE == right_rank ) dest = left_rank  ;
        MPI_Send( &sendbuf, MSIZE, MPI_BYTE, dest, TAG0, COM);
    }
    MPI_Recv( &recvbuf, MSIZE, MPI_BYTE, MPI_ANY_SOURCE, TAG0, COM, &status);
    MPI_Recv( &recvbuf, MSIZE, MPI_BYTE, MPI_ANY_SOURCE, TAG0, COM, &status);
    MPI_Finalize();
    return 0;
}
```

図 3　MPI プログラム(左)および MGEN プログラム(右)

4. MGEN API および MGEN プログラム

本節では，NSIM で通信パターンを記述する際に利用する MGEN API について説明し，MGEN プログラムのサンプルを示す．

4.1 MGEN API と MGEN プログラムの概要

MGEN API は NSIM の MGEN プログラムにおいてメッセージパッシング方式の通信パターンを記述するための API である．MPI で定義されている基本的な宣言については MGEN API でもサポートされている．ただし，集団通信については未サポートであるため，代わりに種々のアルゴリズムを 1 対 1 通信により実装したコードを添付している. MGEN プログラムは MGEN API を利用して記述されたメッセージパッシング方式の通信プログラムである．以降，MGEN API および MGEN プログラムが，通常の MPI や MPI を利用して記述されたプログラムと異なる点について説明する．

MGEN API が MPI と異なる点は，主に 3 点ある．1 点目は，宣言のプレフィクスが“MPI_”ではなく“MGEN_”である点である．これは，MPI プログラムを NSIM でシミュレーションする際に書き換えが必要となることを意味するが，プレフィクス置き換えを行うヘッダファイルを利用することにより, include 行の書き換えのみで MPI プログラムをそのままシミュレーションできる．2 点目は，MPI には存在しない拡張関数が用意されており，この関数を利用することにより複数 NIC による通信，ランデブー通信，パケット間ギャップを指定した通信などを記述することができる点である．さらに, シミュレーション対象システムのノードの CPU 時刻を進める関数が用意されており，ロードインバランスのシミュレーションが可能である．3 点目は，MGEN API で記述された通信では，実際のデータ転送が行われないことである．このため，NSIM では受信したデータに基づき動作を決定する並列プログラムは正しくシミュレーションすることができない．通信によってやりとりしなければならないデータは，あらかじめ各プロセスが参照できるようにプログラムを書き換える必要がある．しかしながら，この制約を設けることにより，シミュレーション対象システムにおけるプロセス間の通信をする必要がなくなるため，消費メモリ量とシミュレーション時間を大幅に削減することができる．

MGEN プログラムが MPI プログラムと異なる点は, MGEN API の呼び出し箇所以外の部分(計算部分)の処理時間がすべて無視される点である．すなわち，計算部分の処理時間を含めてシミュレーションするためには，対象システムにおける計算処理時間を見積もり，CPU 時刻を進める関数を追加してノードの CPU 時刻を進める必要がある．

4.2 MGEN プログラム例

図 3 に双方向リング通信の MPI プログラムと，MGEN API を利用した MGEN プログラムを示す．MGEN プログラムは，MGEN API を利用するためにヘッダファイルとして mpi.h の代わりに mgen-mpi.h をインクルードするように書き換えてある(図 3 赤字部分)．このヘッダファイルは MPI を MGEN API へ置き換えるため，図 3 の例ではその他の変更をすることなく NSIM でシミュレーションすることができる．

5. NSIM の実装

NSIM は，並列離散事象シミュレーション方式で実装している．シミュレーション対象システムにおける各プロセスの時刻は，保守的アルゴリズムに基づき同期している．並列離散事象シミュレーションにおける時刻同期方式には，保守的アルゴリズムと楽観的アルゴリズムと呼ばれる 2 種類の方式が知られている．これらの違いは，保守的アルゴリズムでは投機的実行を許さないが，楽観的アルゴリズムでは投機的実行を許すという点である．このため，楽観的アルゴリズムでは，投機的実行が失敗した場合に再実行するためのロールバック機構が必要であるが，保守的アルゴリズムではこの機構は不要である．したがって，実装は保守的アルゴリズムの方が容易である．シミュレーション速度は，保守的アルゴリズムより時刻同期回数が少ない楽観的アルゴリズムが高速となる可能性があるが，投機的実行結果が間違った場合のロールバックや再実行のオーバヘッドを含めると，いずれかの方式が常に高速であるとは言えない．

我々は，NSIM をメモリ共有並列計算機だけでなく，メモリ非共有計算機での動作に対応させるため，MPI を利用したメッセージパッシング方式の並列プログラムとして実装した．メモリ共有およびメモリ非共有計算機で動作させるための実装方法としては，MPI のみによる実装のほか，MPI とスレッド並列のハイブリッドによる実装が挙がる．後者は前者より性能面では有利であるが，実装およびデバッグが難しくなる懸念がある．前者は実装が容易であるが，プロセス間の通信が多い場合，後者との性能差が開く可能性がある．我々は，実装およびデバッグの容易性を維持しつつ，高いシミュレーション性能を得るため，MPI のみで実装することとした上で，MPI 通信オーバヘッドの削減方法を検討した．離散事象シミュレーションを MPI のみで実装した場合，プロセス間での通信が発生するのは，時刻同期時，およびイベント情報の他プロセスへの移動時，の 2 つがある．このうち，時刻同期については，現在時刻を基準に，新たなイベントの発生がないことを保証できる時間幅を求め，その間の時刻同期を省略することで MPI 集団通信回数を削減できる．また，イベント情報の移動時には，複数のイベントを 1 まとまりにして転送することにより MPI 通信オーバヘッドを削減することができる．NSIM ではこれまでに前者のオーバヘッド削減策を取り入れることで，通信オーバヘッドを低減している．

表 1　シミュレータ実行環境

項目	内容
計算機	Dell Precision™ T7400 workstation [11]
CPU	クアッドコア Intel® Xeon® CPU E5440 (2.8GHz 動作，デュアル 6MB L2 cache を共有) 2 ソケット
主記憶	32GB 4 チャネル DDR2-SDRAM (667MHz 動作，ECC 付き)
OS	Red Hat® Enterprise Linux® 5 Update 4 kernel-2.6.18-164.11.1.el5
MPI ライブラリ	MPICH2 version 1.2 [10]
C コンパイラ	GNU compiler collection version 4.1.2 20080704 (Red Hat 4.1.2-46)

6. 実験

6.1 シミュレーション環境

表 1 にシミュレータ実行環境を示す．この計算機はクアッドコアプロセッサを 2 基搭載したワークステーションであり，合計 8 個のコアで 32GB の物理メモリを共有している．以降の実験では，このワークステーション 1 台を利用して NSIM を実行する．

6.2 シミュレーション精度に関する考察

本節では，実機で観測された結果と NSIM で予測した結果とを比較し，シミュレーション精度について考察する．ベンチマークプログラムは HPC Challenge [12]のランダムリングトラフィックとした．実機で測定された結果として，HPC Challenge に提出されている Intel® Endeavor の結果を参照する．公開されている情報によると，このマシンは Infiniband 8X QDR に対応した Mellanox® MTS3600Q-1UNC スイッチ(36 ポート)を利用してノードを接続している．各ノードは CPU としてクアッドコアの Intel® Xeon® X5560 (2.8GHz 動作)，HCA として Mellanox® MHGH-28XTC を搭載しているとのことである．

実機におけるランダムリングトラフィックを NSIM でシミュレーションするため，評価対象システムの性能や構成に関するパラメータを公開されている仕様などから推定した．表 3 に対象システムの仕様の見積もりを示す．トポロジは，実機で Infiniband が採用されており，FBB (Full Bisectional Bandwidth) 構成になっている可能性があるこ

Dell Precision は，米国 Dell Inc.の商標または登録商標です．Intel および Xeon は，アメリカ合衆国およびその他の国における Intel Corporation の商標です．Linux は米国及びその他の国における Linus Torvalds の登録商標です．Red Hat は，米国 Red Hat, Inc. ならびにその子会社の登録商標です．Mellanox は Mellanox Technologies, Ltd.の登録商標です．

表 2　評価対象システムのパラメータ見積もり

パラメータ	設定値
相互結合網仕様	
トポロジ	FatTree (Infiniband)
単方向リンクスループット	4GB/s
スイッチバンド幅	4GB/s
ルーティング計算時間	10ns
仮想チャネルアロケーション時間	10ns
スイッチアロケーション時間	10ns
スイッチ遅延時間	140ns
ケーブル遅延時間	200ns
MTU サイズ	2048B
ノード設定	
HCA 転送レート	2GB/s (1 ポート)
DMA 転送レート	10GB/s
メモリバンド幅	10GB/s
MPI 関数処理オーバヘッド	200ns

とから，NSIM でも FatTree の FBB 構成とした．ルーティングアルゴリズムは実機のアルゴリズムが不明であったため，NSIM では上り直進ルーティングを選択している．MGEN プログラムはベンチマークプログラムのコードからランダムリングトラフィック通信に関係する箇所を切り出して作成した．この MGEN プログラムを NSIM でシミュレーションし，ベンチマークプログラムと同様の算出方法でスループットを求めた．

図 4 に結果を示す．グラフの横軸がノード数，縦軸がスループット(単位は GB/s)である．NSIM による予測結果は赤，実機での観測結果は青で表している．両者は，ノード数の増加とともに同様の傾向で変化している．しかしながら，NSIM での予測結果は実機での測定結果と比較して常に高いスループットを示している．この原因としては，ノード内メモリスループットの違いなど，実機での様々なオーバヘッドを NSIM では理想化している点が挙がる．

6.3 シミュレーション性能に関する考察

ワークステーションなどの並列計算機上において，数 10 万ノード規模の相互結合網のシミュレーションが可能であることを示すため，NSIM のシミュレーション時間と消費メモリ量を観測した．比較のため，既存のシミュレータである BigNetSim につ

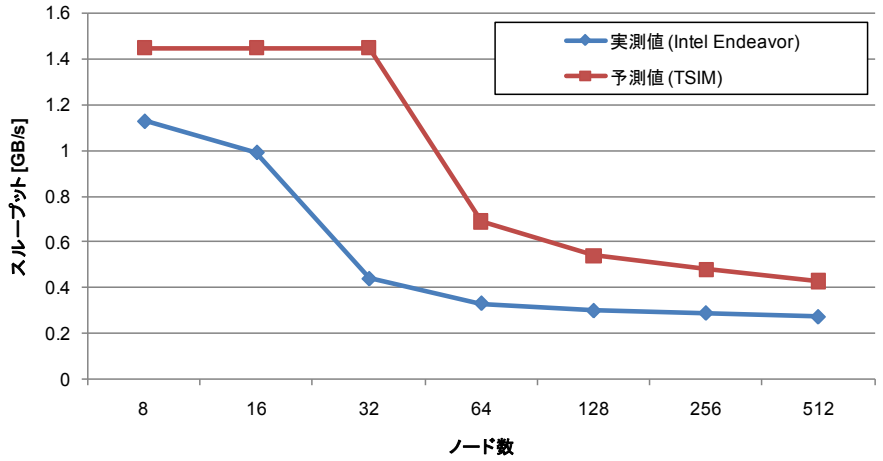

図 4　NSIM によるバンド幅予測値(赤)と実機での観測結果(青)

いても同様にワークステーション上でシミュレーションした．BigNetSim は Rev.11877，Charm++ライブラリ(BigSim)は BigNetSim に対応するバージョンをリポジトリから取得した．

シミュレーション対象システムの各パラメータは表 3 の値を利用する．これらの値は，シミュレーション対象システムとして将来の相互結合網を想定し，表 2 の値を参考にして決定した．シミュレーション対象の相互結合網のトポロジは 3 次元トーラス網とし，ノード数は 8 ノードから 64K ノードまで変化させた．なお，BigSim および BigNetSim では BlueGene/L の相互結合網モデルを選択した．ベンチマークプログラムは，ネットワーク負荷が高い全体全通信(Bruck のアルゴリズム[8])を利用した．このアルゴリズムは，転送ステップ数が他の全体全通信アルゴリズムより少ないが，各プロセスの転送データ量が大きいという特徴があり，転送データサイズのオーダは O(n·log(n))と表される．このアルゴリズムを NSIM 向けおよび BigNetSim 向けに MGEN プログラムおよび MPI プログラムとして実装した．なお，NSIM では MGEN プログラム内の不要な呼び出しを削減するため，MGEN_Key 関数の呼び出しを追加している．メッセージ長は 4Byte および 1024Byte の 2 種類とする．

各シミュレータを 8 つのコアを使って動作させる．シミュレーション時間はシミュレーション完了時に各シミュレータが出力する値を利用する．ただし，BigSim のシミ

Vol.2010-HPC-125 No.5
2010/6/17

表 3　NSIM，BigNetSim における評価対象システムの設定

パラメータ	設定値
相互結合網仕様	
トポロジ	3 次元トーラス網
単方向リンクスループット	8GB/s
スイッチバンド幅	8GB/s
ルーティング計算時間	2ns
仮想チャネルアロケーション時間	2ns
スイッチアロケーション時間	2ns
スイッチ遅延時間	140ns
ケーブル遅延時間	100ns
MTU サイズ	256B
ノード設定	
DMA 転送レート	10GB/s
メモリバンド幅	10GB/s
MPI 関数処理オーバヘッド	200ns

ュレーション時間は比較的小さいため無視することとし，BigNetSim のシミュレーション時間のみを取得した．また，メモリ利用量は ps コマンドを „-o vsz’ 引数付きで 0.01 秒ごと(実行開始から 1 秒まで)もしくは 1 秒ごと(それ以降)に実行し，最大値を算出した．各シミュレータによる予測結果は，実装しているモデルが異なるため一致することはないが，大幅に異ならないことを確認している．

図 5 に，各シミュレータのシミュレーション実行時間およびメモリ消費量を示す．図 5 において，横軸はノード数，左縦軸はシミュレーション実行時間，右縦軸はメモリ消費量を示す．図 5 中で，線グラフがシミュレーション実行時間，棒グラフが消費メモリ量に対応する．NSIM ではメッセージサイズ 4 バイトで 64K ノード(64x32x32，4 バイトメッセージ)および 4096 ノード (16x16x16，1024 バイトメッセージ)までのデータを取得した．BigNetSim では，シミュレーション環境に搭載されている物理メモリ量制約のため，最大で 1024 ノード(4 バイトメッセージ)および 128 ノード(1024 バイトメッセージ)までの結果を取得することができた．

まず，シミュレーション実行時間について考察する．NSIM のシミュレーション実行時間は，4 バイトメッセージの場合に 7 時間弱(64K ノード)，1024 バイトメッセージの場合に 1 時間半弱(4K ノード)となった．今回のシミュレーション実行環境では，NSIM は BigNetSim と比較して 2700 倍(1024 ノード，4 バイトメッセージ)，370 倍(128

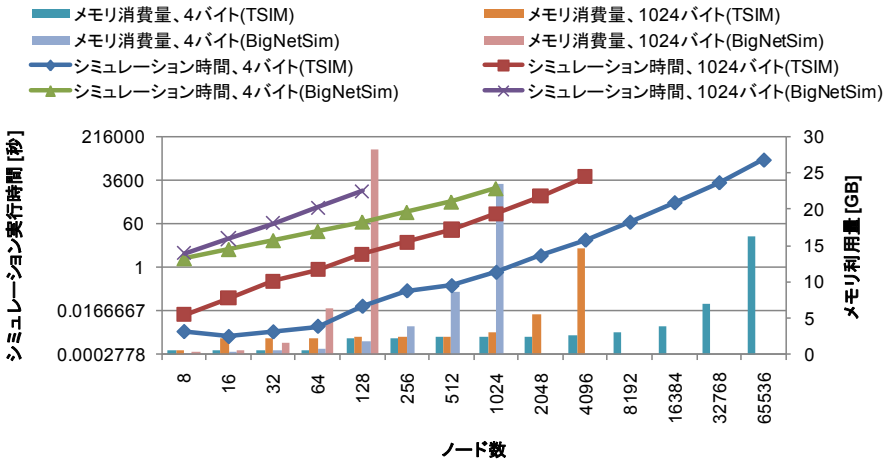

図 5　NSIM および BigNetSim のシミュレーション実行時間および最大メモリ消費量

ノード，1024 バイトメッセージ)高速に動作することが確認できる．以上から，メッセージサイズが小さい場合，NSIM は 10 万ノードに近い規模のシミュレーションを現実的な時間内に完了させるシミュレーション性能を持っていることを確認できる．

次に，各シミュレータの最大メモリ消費量について考察する．1024 ノードで 4 バイトメッセージを転送する際，BigNetSim では 23GB のメモリを必要としているが，NSIM では 2.3GB と 1/10 で済んでいる．このことから，NSIM は BigNetSim と比較して省メモリであることを確認できる．さらに，NSIM において 64K ノードで 4 バイトを転送する場合のメモリ消費量は 16GB となった．したがって，NSIM は総搭載物理メモリ容量が少ない計算機においても大規模なシミュレーションを行う能力があると言える．

以上の結果から，NSIM は 10 万ノード規模の相互結合網のシミュレーションをデスクサイドのワークステーションのような，並列度が低いメモリ共有計算機で実現する能力を持っていることを確認できる．このことは，NSIM のユーザに新たなメリットを提供できることを示している．手元の計算機でシミュレーションをしながらログの処理をすることにより，シミュレーション完了前にシミュレーション対象システムにおける輻輳発生状況をユーザが把握することが可能となる．大規模相互結合網のシミュレーション時間は長くなる傾向にあることから，シミュレーション中にそのような情報をユーザに提供することにより性能評価および解析期間を減らすことができる．

7. おわりに

本稿では，将来の相互結合網の性能を予測するための相互結合網シミュレータ NSIM を紹介した．NSIM では，実行駆動方式を採用することで輻輳やロードインバランスなどに起因して振る舞いが変化する通信パターンのシミュレーションを正しく行うことができる．また，対象システムにおけるデータ転送ののシミュレーションを省くことでシミュレーション時間およびメモリ消費量を削減している．このため，ワークステーション等の小規模なメモリ共有計算機上でも，マルチコア・プロセッサの低い通信レイテンシを利用して大規模相互結合網のシミュレーションを高速に行うことができる．NSIM によるシミュレーション結果と実機の結果の比較により，スループットは類似の結果が得られていることから，本シミュレータによる予測結果は定性的に正しいことを確認した．また，クアッドコアプロセッサを 2 基搭載したワークステーション上でのシミュレーション規模の調査から，現実的な時間内で 10 万ノード規模のシミュレーションを完了する能力を持つことを示した．

NSIM の課題は以下の通りである．1 点目は，精度検証を進め，必要に応じてルータおよびノードのモデルを詳細化することである．これは，NSIM に実装しているモデルは特定のハードウェアの詳細仕様を参考にしたものではなく，抽象化もしくは省略している箇所が存在するためである．いくつかのルーティングアルゴリズムおよび調停アルゴリズムの実装も必要である．2 点目は，我々が以前提案した BSIM [9]の枠組みにおいて NSIM を活用し，将来の大規模並列計算機システムにおけるアプリケーション性能評価可能性を検証することである．大規模並列計算機におけるアプリケーション性能予測を目的とする BSIM と，大規模相互結合網における通信パターンの性能評価を目的とする NSIM との協調シミュレーションにより，精度の良い性能予測結果が得られることを示す予定である．

謝辞 本シミュレータの開発，および本稿の執筆にご協力下さった，九州先端科学技術研究所ならびに九州大学の諸氏に謹んで感謝の意を表します．

参考文献

1) N. R. Adiga, M. A. Blumrich, D. Chen, P. Coteus, A. Gara, M. E. Giampapa, P. Heidelberger, S. Singh, B. D. Steinmacher-Burow, T. Takken, M. Tsao, and P. Vranas, “Blue Gene/L torus interconnection network,” IBM Journal of Research & Development, Vol. 49, No. 2/3, pp.265–276, 2005.
2) D. M. Nicol, C. C. Micheal, P. Inouye, “Efficient Aggregation of Multiple LPs in Distributed Memory Parallel Simulations,” Proceedings of the Winter Simulation Conference, Dec. 1989, pp. 680-685.
3) N. Choudhury, T. Mehta, T. L. Wilmarth, E. J. Bohm, and L. V. Kale, “Scaling an optimistic parallel simulation of large-scale interconnection networks,” Proc. of the Winter Simulation Conference, pp. 4-7, Dec 2005.
4) G. Zheng, G. Kakulapati, L. V. Kale, “BigSim: A Parallel Simulator for Performance Prediction of Extremely Large Parallel Machines,” Parallel and Distributed Processing Symposium, International, Vol. 1, No. 1, pp. 78b, 18th International Parallel and Distributed Processing Symposium (IPDPS’04) - Papers, 2004.
5) T. L.Wilmarth, L. V. Kale, “POSE: getting over grainsize in parallel discrete event simulation,” International Conference on Parallel Processing (ICPP), pp. 12-19, Aug. 2004.
6) L. V. Kale and S. Krishnan, “CHARM++: A Portable Concurrent Object Oriented System Based On C++,” In Proceedings of the Conference on Object Oriented Programming Systems, Languages and Applications, pp. 91-108, 1993.
7) Y. Ajima, S. Sumimoto, T. Shimizu, "Tofu: A 6D Mesh/Torus Interconnect for Exascale Computers," IEEE Computer, pp. 36-40, November, 2009.
8) J. Bruck, C. Ho, S. Kipnis, and D. Weathersby, “Efficient algorithms for all-to-all communications in multi-port message-passing systems,” Proceedings of the Sixth Annual ACM Symposium on Parallel Algorithms and Architectures, pp.298-309, 1994.
9) R. Susukita, H. Ando, M. Aoyagi, H. Honda, Y. Inadomi, K. Inoue, S. Ishizuki, Y. Kimura, H. Komatsu, M. Kurokawa, K. J. Murakami, H. Shibamura, S. Yamamura, Y. Yu, “Performance Prediction of Large-scale Parallel System and Application using Macro-level Simulation,” Proc. of Supercomputing (SC2008), Nov. 2008.
10) MPICH2, http://www.mcs.anl.gov/research/projects/mpich2/
11) Dell Precision TM T7400, http://support.dell.com/support/edocs/systems/wsT7400/
12) HPC Challenge Benchmark, http://icl.cs.utk.edu/hpcc/