Vol.2015-HPC-151 No.7
2015/9/30

スライド手法によるスペアノードを利用した故障復帰方式の性能評価

吉永 一美[1,a)]　亀山 豊久[1]　堀 敦史[1]　石川 裕[1]

概要：エクサスケール環境ではシステムの MTBF が短縮されるために，耐故障性の確保が重要な課題である．我々は，メッシュ/トーラストポロジネットワーク上でのステンシル計算を対象とした故障復帰手法としてスペアノードを利用した手法を提案し，研究を進めている．スペアノードを利用することでロードバランスを保ち，論理的な通信パターンも維持した実行継続が可能であるが，スペアノードをどのように利用して故障ノードを代替させるかが通信性能に大きく影響を与える．そこで我々はスペアノードを利用する手法としてスライド手法を提案し，その評価を進めている．本稿では，3 次元メッシュ/トーラストポロジーのネットワーク上において，7 点ステンシル計算を対象とし，スライド手法により故障復帰を行った際の通信性能について評価を行う．同時に，故障ノード数が増加した場合の性能についても評価を進める．そして実環境におけるスライド手法による故障復帰後の通信性能について，実験を行い議論する．

1. はじめに

システムを構成するハードウェア数が増大するエクサスケール環境では，故障発生率が増加するため，システムの MTBF(平均故障間隔/Mean Time Before Failure) の短縮が懸念されている．現状の HPC 環境にて広く応用されている耐故障手法であるシステムレベルでのチェックポイント手法は，システムによりアプリケーション実行時に定期的にチェックポイントをストレージへと保存し，そのチェックポイントから状態を復元することで耐故障性を実現するものである．しかし規模の増大するエクサにおいては，チェックポイントのデータ量が大幅に増加し，I/O に要する時間が長時間となるために，実行時間の多くをチェックポイント処理に費やしてしまい，現実的ではない [1]．このような背景から，アルゴリズムレベルで耐故障性を実現する手法が多く提案されている．この手法により，チェックポイントとして実行再開に必要なデータのみを保存するなどによる I/O 時間の短縮や，故障発生時の対応をアプリケーション内に定義することにより効率的な実行継続などが実現可能となる．しかしながら，現行の MPI ライブラリの規格 [2] では，故障発生時の挙動が定義されていないために，アルゴリズムレベルでの耐故障性は実現不可能である．

ULFM(User Level Failure Mitigation) [3], [4] は，この問題を解決しアルゴリズムレベルでの耐故障性を実現可能とする MPI ライブラリである．ULFM では故障発生時の挙動を定義しており，実行中にノード故障が発生した場合に，プログラムでの故障検知を実現するとともに，停止プロセスを除いた新たなコミュニケータの生成などの，故障対処のための API を提供している．この ULFM は次期 MPI の標準規格となる見通しである．また，ULFM はアプリケーションに耐故障性を実現するための最低限の機能を提供するものであるため，ULFM を利用した上位のフレームワークも研究開発が進められている [5], [6].

以上のように，エクサにおいて耐故障性を実現する技術の研究は多くなされている．しかし一方で，その研究成果を用いてどのように実行を継続することが効果的であるかについては，現状ではあまり研究がなされていない．

我々は，ULFM のようなライブラリを用いて，ノード故障後のジョブの継続が実現可能となった場合に，どのように故障復帰することが効率的であるかについて研究を進めている [7], [8], [9], [10], [11]．これまでに，科学技術計算分野にて広く使われているステンシル計算アプリケーションを対象とし，スペアノードを利用した故障復帰手法の提案と評価を進めてきた．スペアノードを利用することでアプリケーション・プログラムに大きな変更を加えること無く，ロードバランスを崩さずに実行継続が可能となる．その一方でネットワーク的に離れた位置にあるスペアノードを用いることで，通信衝突が発生し，通信性能が低下することを確認している．そこで，スペアノードの利用方式と

[1] 理化学研究所 計算科学研究機構
RIKEN AICS
[a)] kazumi.yoshinaga@riken.jp

して 3 種のスライド手法を提案し，基礎研究として 2 次元メッシュネットワーク上での 5 点ステンシル計算を用い，評価を進めてきた．

本論文では，我々の提案するスライド手法によるスペアノードを利用した故障復帰について，3 次元メッシュネットワークと 7 点ステンシル計算を対象とした評価を行う．また，これまでは数ノードまでしか評価しなかった故障台数についても，増加することでどのように性能が変化するのか検証を行う．以降，まず第 2 章において，スペアノードを用いた故障復帰について，その有用性や特徴を述べる．そして第 3 章において，スペアノードの割り当て方式であるスライド手法について，各手法の解説を行う．次に第 4 章で，シミュレーション及び実験の結果を示し，スライド手法の評価を行う．そして第 5 章でまとめを行う．

2. スペアノードを用いた故障復帰

2.1 スペアノードの有用性

あるノードに故障が発生した場合の故障復帰手法として，故障ノード上で行われていた計算を，別の正常なノードに割り振ることによってジョブの継続を行う手法がある．我々はこの手法を「しわ寄せ法」と呼んでいる．しわ寄せ法による実行継続は，マスターワーカモデルのようなロードバランスを動的に保つ機能を持つジョブに対しては有効である．しかしながら，本稿で対象としているステンシル計算についてはしわ寄せ法は不適当である．ステンシル計算は問題サイズに応じて適切なノード数を設定し，各ノードに計算領域をバランスして割り付ける事によってロードバランスを実現している．このため，故障する毎にノード数が変化するしわ寄せ法による故障復帰では，ロードバランスを保った計算の割り当てが困難である．更に，適切にロードバランスを保つように問題の再分割を行ったとしても，通信パターンに変化が生じてしまう．ステンシル計算は隣接領域を計算するノードとのみ通信を行う単純な通信パターンを持つが，しわ寄せ法による実行継続を行った場合問題領域の分割が複雑になり，この単純な通信パターンを保つことが出来ない．そのため，故障復帰に対応したプログラムを作成するためには，故障後の多岐にわたる複雑な通信パターンに対応したコードを記述する必要があり，プログラマの負担を増大させると共にバグの混入を招きやすくなる．

これらの問題を解決することのできる故障復帰手法が，スペアノードを利用した手法である．故障ノード上で実行されていた計算をスペアノードに移動して実行継続を行うことで，ロードバランスを崩さない故障復帰手法が容易に実現できる．また，故障ノード上で実行されていたプロセスの実行ノードが変化するだけであるため，プログラム上では通信パターンも崩れることはない．例えば MPI プログラムであれば，故障前はスペアノードを除いたノード群で構成される MPI コミュニケータを作成し，それを用いて実行する．故障後は故障ノードを除き，代替するスペアノードを含めた新たな MPI コミュニケータを作成して置換するだけで，計算や通信といったカーネルコードの変更をすることなく実行継続が可能となる．

2.2 スペアノードの確保方式

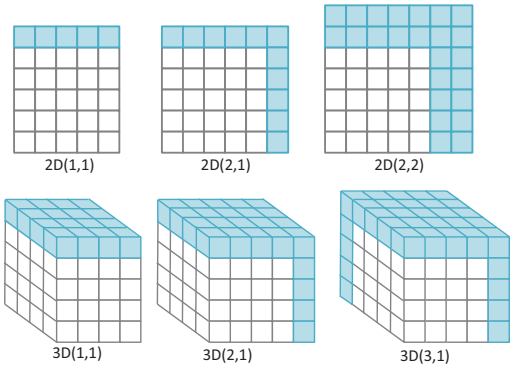


図 1 スペアノード確保例

図 1 に，2D，3D ネットワークでのスペアノードの確保例を示す．この図において，nD(α,β) がスペアノードの確保方式を示しており，n がネットワークの次元数を，α がスペアノードを確保する次元 (辺または面) の数を，β が厚さを示している．

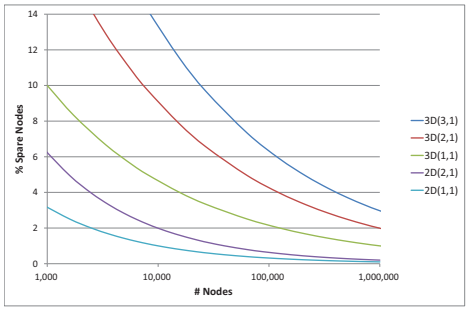


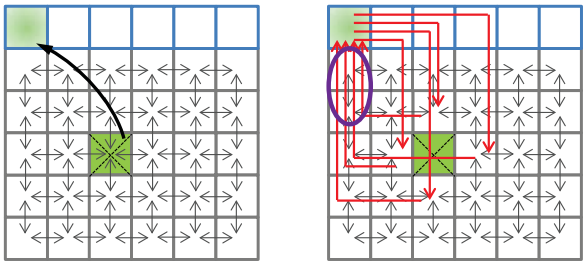
図 2 スペアノードの占める割合

図 2 に，ジョブ実行の為に確保したノードに対してスペアノードが占める割合を計算式により求めた結果を示す．なお，2D では正方形形状で，3D では立方体系上でノードを確保した場合を仮定している．スペアノードの確保方式は前述した通りである．

スペアノードの次元数はネットワークトポロジの次元数よりも必ず少ないため，ノード数が増加するに伴いスペアノードの割合が低下する．図 2 より，ノード数が少ない場合はスペアノードの確保によるオーバーヘッドは大きいが，10 万ノード規模になればスペアノードの割合は数パーセント程度になることがわかる．

2.3 スペアノード利用による通信性能への影響

スペアノードを利用することで，プログラム内での論理的な通信パターンに変化が無いことは 2.1 節で述べたとおりである．しかし，プロセスの実行ノードが変化するために，物理的な通信パターンは故障の前後で異なるものとなり，通信遅延の増大や，通信衝突の発生による通信時間の増加が発生する可能性がある．



(a) 故障発生までの通信パターン　(b) 故障復帰後の通信パターン

図 3 スペアノード利用による通信衝突の発生

図 3 に，2D メッシュネットワーク上での 5 点ステンシル計算を例とした，スペアノードを利用した故障復帰により発生する通信衝突を示す．**図 3(a)** は故障発生前の通信パターンである．5 点ステンシル計算の通信パターンは，隣接するプロセスとの通信であり，ジョブ実行時に適切にノード割り付けを行えば，衝突無く通信することが可能である．ここで，×で示したノードが故障し，スペアノードを利用して故障復帰した例が**図 3(b)** である．論理的には隣接プロセス間通信のみという通信パターンを保っているが，スペアノードを利用した結果，物理的な通信パターンが変化し，○で囲んだ部分において通信衝突が発生している．通信衝突の発生を抑制するためには，スペアノードをどのように利用し，故障ノードと代替するかが重要となる．次章において，スペアノードの利用手法として我々が提案しているスライド手法について解説する．

3. スライド手法によるスペアノードの利用

本章では，スペアノードの利用方式として提案するスライド手法について，基本となる 0D，1D，2D，3D Sliding の 4 種のスライド手法と，その組み合わせによる Hybrid Sliding について説明する．なお，**図 4** に示す 0D,1D,2D Sliding の 3 種については，[8], [9], [10] にて詳細に示しているため，簡単な説明に留める．

3.1 0D Sliding

0D Sliding 手法は，図 4 の右上のように，故障ノード上で実行していたプロセスを単純にスペアノードへと移動することで，故障復帰を行う手法である．この手法の特徴として，スペアノードの台数までの故障に対応可能であるこ

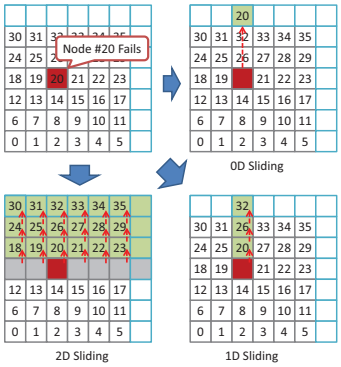


図 4 0D,1D,2D Sliding 手法 (2D メッシュネットワーク)

とと，離れたノードへと移動するため増加するホップ数が多く，通信衝突が増加しやすいことが挙げられる．通信衝突の増加を低減させるためには，代替するスペアノードの位置の選択が重要である．3D ネットワーク上においても，[8] で述べた選択順を踏襲し，故障ノードと出来る限り近い軸上にあるスペアノードを優先的に利用する．

3.2 1D Sliding

1D Sliding 手法は，故障ノードが発生した際に，故障ノードと同一の軸 (2 次元であれば同一列または行，3 次元であれば x，y，z 座標のうちいずれかが同一) であるスペアノードを選び，故障ノードからスペアノードまでの間にあるプロセス全てを，スペアノードに向けて 1 ノードずつスライドさせる方式である．1D Sliding 適用時に増加する通信ホップ数は最大 1 であるため，0D Sliding と比較して発生する通信衝突数が少なくなり，性能低下が少ない．

3.3 2D Sliding

2D Sliding 手法は，2D メッシュ/トーラスネットワークであれば，故障ノードと同一行または列に存在する全てのノードを使わないように，2 次元でスライドする手法である．2D メッシュ/トーラスネットワークであれば，本手法での故障復帰により通信衝突は発生しない．

3D メッシュ/トーラスネットワークにおける 2D Sliding も，同様に 2 次元平面上で一括してスライドするものである．ただし 3D メッシュ/トーラスネットワークで本手法を適用した場合は，通信衝突が発生する．2D Sliding 手法も 1D Sliding と同じく増加する通信ホップ数は最大で 1 であるが，ズレの生じる次元数が 1D と比較すると 1 次元小さくなるため，ホップ数の増加する通信数が少なく，通信衝突数も少ない．

3.4 3D Sliding

3D メッシュ/トーラスネットワークトポロジにおいては，通信衝突の発生しないスライド手法として 3D Sliding 手法が考えられる．この手法では，故障ノードと同一の x

座標 (または y 座標か z 座標) を持つ全てのノードを利用しないように，3 次元的に一括スライドする．**図 5** に，3D Sliding 手法の適用例を示す．本手法では通信衝突は発生しない効率的な故障復帰を実現する一方で，対応可能な故障台数は少なく，3D(1,1)，3D(2,1)，3D(3,1) の場合において，1 台，2 台，3 台がそれぞれ上限となる．

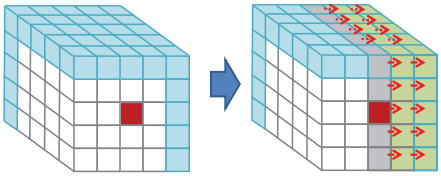


図 5 3D Sliding 手法

3.4.1 Hybrid Sliding

これまでに述べてきたスライド手法のうち，2D 及び 3D Sliding 手法では，故障ノードだけでなく生存しているノード上のプロセスも移動する．そのためこれらの手法を用いた場合，故障台数に対するスペアノードの利用台数が大きくなり，対応可能な故障台数が 0D や 1D と比較して大きく低下する．また，1D 及び 2D Sliding 手法では，スペアノードがまだ残っている状況においても，故障ノードと残りスペアノードの位置関係によっては，実行の継続が不可能になる場合がある．

これらの問題を解決するために考案したのが，各手法を組み合わせた Hybid Sliding 手法である．Hybrid Sliding 手法では，2D，3D Sliding 手法の適用により生存しているにも関わらず空きとなったノードを，スペアノードとして再利用する．高次元のスライド手法では実行継続が出来なくなった場合，残りのスペアノードを利用し，適用可能な低次元のスライド手法により故障復帰を行うことで，対応可能な故障台数を増加させることができる．

Hybrid Sliding は，手法の組み合わせ方により複数種考えることができる．本論文では，全てのスライド手法を組み合わせたものを単純に Hybrid 手法と呼び，これ以外のものは適用する手法を () 内に示すことで区別する．例えば，まず 3D Sliding の適用を最初に試み，それが不可能であれば 0D Sliding を適用する手法は Hybrid(3D+0D) のように示す．

4. スライド手法の性能評価

4.1 シミュレーションによる評価

スライド手法により故障復帰を行い，ジョブの実行を継続した際の性能を確認するために，まずは各方式を利用した実行継続をシミュレーションするプログラムを作成し，その通信衝突について調査を行った．

このシミュレーションでは，3D メッシュネットワークを仮定し，その上で 7 点ステンシル通信を行う．そしてノード故障をランダムに発生させ，提案した各方式による故障復帰をシミュレートし，得られたノード配置を用いて通信を行った際の最大通信衝突数を求める．この最大通信衝突数をパラメータとして評価を行う．

シミュレーションの実施条件は，表 1 に示す通りである．本シミュレーションにおいて，ステンシル通信は非周期境界を仮定する．また，故障ノードはランダムに決定していくため，スペアノードを含む空きノードが故障する場合も存在している．

表 1 シミュレーション条件

ノード形状	12x12x12
スペア確保方式	3D(2,1)
最大故障数	276(スペアノード数と同数)
検証故障パターン	3,686,400

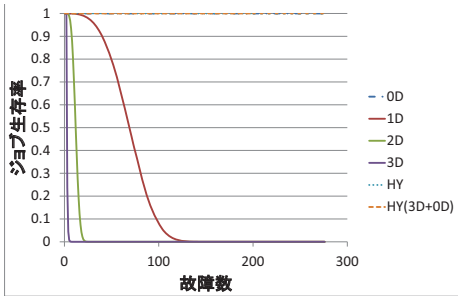


図 6 各方式のジョブ生存率

図 6 は，ノード故障発生後のジョブ生存率を手法別に示したものである．この図より，低次元なスライド手法ほどジョブの生存率が高いことが見て取れる．0D Sliding 及び 2 種の Hybrid Sliding については，スペアノードを全て利用できるため生存率は常に 100%である．スペアノードを 3D(2,1) で確保しているため，3D Sliding により確実に実行継続可能となるのは，2 ノード故障までである．それ以上の故障数で 3D Sliding が生存しているパターンは，空きノードが故障ノードとして設定された場合である．

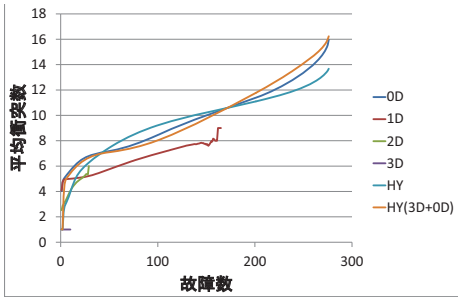


図 7 各手法の平均メッセージ衝突数

図 7 は，各手法を用いて故障復帰を行った後のメッセージ衝突数について，実験した故障パターンでの平均値を示したものである．同様に，**図 8** は求まった値の最悪値を，

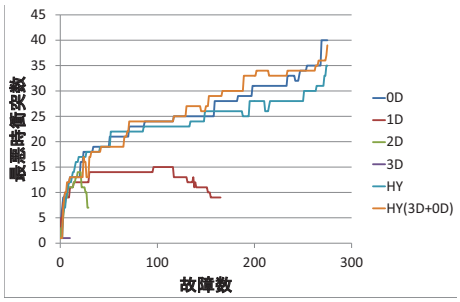


図 8 各手法の最悪時メッセージ衝突数

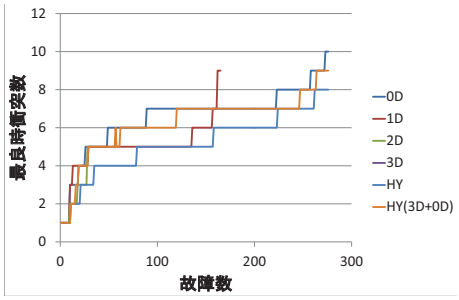


図 9 各手法の最良時メッセージ衝突数

図 9 は最良値を示したものである．

故障台数が増加するにつれて 1D Sliding 手法の性能の変動が大きくなっているのは，生存するパターン数が少なくなっているために，1 つの故障パターンあたりのシミュレーション結果の影響が増大しているためである．また，図 8 に示される最悪時の衝突数が 120 ノード故障以降で低下している原因は，衝突数の増加するような故障パターンの多くは故障台数が増加すると生存できなくなり，衝突数の少ない故障パターンのみが残っているためである．

続いて，0D，Hybrid，Hybrid(3D+0D) の 3 手法を比較する．平均衝突数，最悪時衝突数共に，約 170 ノード故障以降のスペアノードの多くを利用する状況においては，Hybrid が最も高性能を示している．しかし，故障ノード数が 50〜170 の間では，平均衝突数を見ると Hybrid 手法が最も性能が悪化していることを確認した．これは，2D Sliding や 1D Sliding を利用することで多くのプロセスが入り乱れて移動することとなり，結果的に単純に置換する 0D Sliding よりも通信衝突が多く発生したためだと考えている．

276 ノードが故障した場合，最も高性能な Hybrid 手法を用いても最悪時で 35，平均でも約 14 の通信衝突が発生している．通信に衝突が発生すると，通信時間が増大する．1 つの通信衝突が発生すると，1 メッセージ分の通信時間増加 (ペナルティ) が発生する．つまり，N 個の通信が衝突すると，単純に考えれば通信時間は N 倍に増大してしまう．このため，故障前のアプリケーションの実行時間のうち通信時間が 10%だったとしても，故障後の実行時間は最悪の場合 4 倍以上に膨れ上がることとなる．この問題を解決するために，ランクの再配置を行うことで通信衝突を軽減する手法が考えられるが，有効な再配置アルゴリズムについてはまだ分かっておらず検討中である．

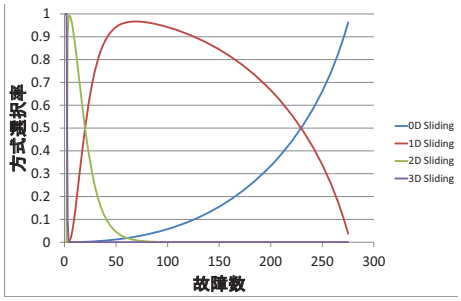


図 10 Hybrid のスライド手法選択率

図 10 は，Hybrid 手法による故障復帰の際に，どのスライド手法が利用されたか，その選択率を示したものである．本シミュレーションでのスペアノード確保手法は 3D(2,1) であるため，2 ノード故障までは必ず 3D Sliding が選択される．その後は 2D → 1D → 0D という優先順で選択していくため，故障ノードが増加するに伴い低次元なスライド手法の選択率が高まっていく．

4.2 実環境における通信性能評価

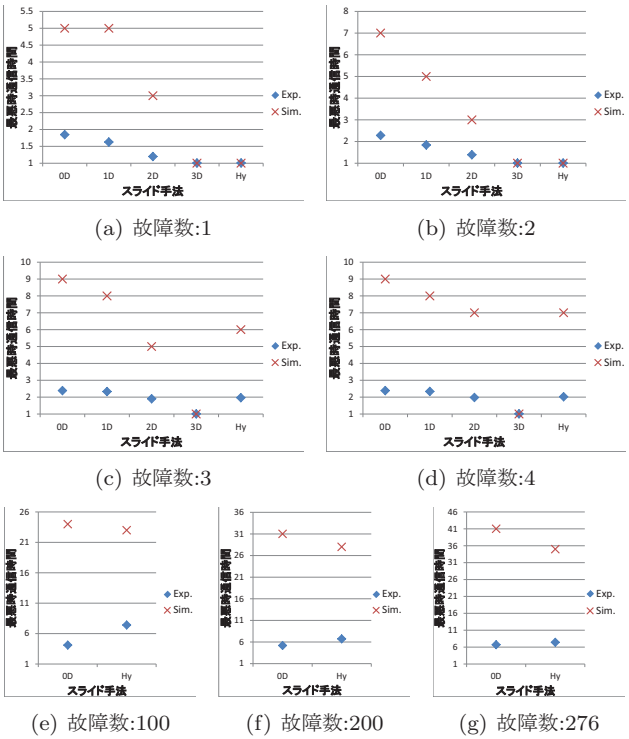
続いて，スライド手法を利用した故障復帰の結果，実環境ではステンシル計算の通信性能にどのような影響が生じるかを，京を用いて実験し，評価を行った．

この実験では，ジョブの実行形状として 12x12x12，スペアノード確保方式として 3D(2,1) を設定する．そして，1D，2D，3D の 3 手法ではノード故障数が 1，2，3，4 の場合について，0D，Hybrid の 2 手法はその 4 個に加えて 100，200，276 故障の場合について，4MiB の 7 点ステンシル通信を行い，通信時間の測定を行う．

実験では，実際に故障を発生させて各手法を適用するのではなく，4.1 節のシミュレーションによって得られた，故障復帰後のノードランクマッピングを用い，模擬的に故障復帰後の状態を再現した上で 7 点ステンシル通信時間の測定を行う．マッピングとしては全シミュレーションパターンを利用するのではなく，スライド手法・故障台数の組み合わせ毎に，メッセージ衝突数が大きかった 768 種を利用する．

図 11 に，手法・故障台数毎に，実験で得られた通信時間の最悪値を示す (項目 Exp.)．縦軸の値は，故障前の状態での通信時間を基準とした倍率である．また，図中には 4.1 節のシミュレーションで得られた通信時間 (＝メッセージ衝突数) も同時に示している (項目 Sim.)．

まず，3D Sliding について着目すると，シミュレーションの結果通り故障後のノードマッピングにおいて通信衝突が発生しておらず，通信性能を維持できていることが確認できる．2 ノード故障までの Hybrid も同様に，図 10 に示



(a) 故障数:1　(b) 故障数:2

(c) 故障数:3　(d) 故障数:4

(e) 故障数:100　(f) 故障数:200　(g) 故障数:276

図 11　各方式による通信性能

した通り 2 ノード故障までは必ず 3D Sliding を利用するため，通信衝突が発生していない．

続いて，4 ノード故障までの残りの結果 (**図 11(a)**〜**図 11(d)**) について着目する．全ての場合において，シミュレーションで得られた予測値よりも通信時間の増加割合が小さくなっており，実測結果はシミュレーション結果と比較して $\frac{1}{3.8}$〜$\frac{1}{2.2}$ 倍にとどまっている．

この差の原因の一つとして，単一方向への 1 つのメッセージ通信時間と，7 点ステンシルで実行される 6 方向同時通信の実行時間の差が挙げられる．1 度のメッセージ衝突が発生することによる通信時間の増大 (ペナルティ) は，1 つのメッセージ通信時間と同一である．そのため，「通信衝突数＝故障前の通信時間との比率」という仮定は，故障前の全 6 方向の通信時間と 1 つのメッセージの通信時間が同一である場合にのみ成り立つものである．実験環境である京の tofu インターコネクトが持つ RDMA エンジンは 4 基であり，6 方向の通信を同時に行うことができない．また，4 基の RDMA エンジンを同時に利用した場合，メモリと RDMA エンジンを結ぶインターフェースの帯域幅に制限され，1 メッセージ通信時と比較して性能が低下する [12]．実際に測定した結果，6 方向通信の通信時間は 1 つのメッセージの通信時間と比較して，3.72 倍となることを確認した．このバイアスを考慮すれば，4 ノード故障までの実測結果とシミュレーション結果の差はおよそ 1〜1.7 倍となり，シミュレーション結果よりも悪化していることになる．この原因としては，3D メッシュネットワーク環境を 6D メッシュトーラスネットワークである tofu にマッピングしているために，スライド時にシミュレーションとは異なる経路を利用した通信が発生している可能性があることや，ハードウェアの制約上 6 通信の発行は同時に行えずに 2 度に分けて行うことになり，通信タイミングがずれたことによりシミュレーションでは再現できなかった通信衝突が発生している可能性があること等が考えられる．

続いて，故障ノード数が 100，200，276 における結果 (**図 11(e)**，**図 11(f)**，**図 11(g)**) に着目すると，シミュレーションでは 0D よりも高性能を示した Hybrid が，全ての故障ノード数において 0D よりも低性能になっている．Hybrid による実測通信時間は，シミュレーション結果からの予測の $\frac{1}{4.7}$〜$\frac{1}{3.1}$ 倍になっているのに対して，0D の場合は全て $\frac{1}{6}$ 程度となっている．前述の 6 方向同時通信時間と 1 メッセージ通信時間の比を考えると，0D はシミュレーションと比較して非常に高性能を示している．この要因もマッピングによる影響が考えられるが，詳細についてはわかっておらず，現在調査中である．

5. まとめ

本論文では，エクサの時代に向けてステンシル計算に対する耐故障性の実現手法として研究を進めている，スペアノードを用いたスライド手法について，3D メッシュ/トーラスネットワーク上での 7 点ステンシル計算に焦点を当て評価を進めた．

3D ネットワークにおけるスライド手法としては，0D，1D，2D，3D Sliding の 4 手法と，それらを組み合わせた手法である Hybrid Sliding 手法を提案し，シミュレーション及び京を用いた実験により評価した．

シミュレーションにより求めた通信衝突数に基づく評価では，今回の実験条件においては，故障数が多い場合は Hybrid 手法が最も高性能を示した．しかし，最大の故障数において，最悪の場合での通信衝突数は 35，平均衝突数でも 14 となっており，無視できない性能低下である．今後何らかの改善を進めていく必要性がある．

京を用いた実験では，全ての場合においてシミュレーションにより当初予測された通信時間よりも短くなるという結果となった．これは，非故障時において，1 メッセージの通信時間と，6 方向通信の通信時間に隔たりがあるためであり，これを考慮して補正を行うと，故障台数が少ない場合は予測よりも低性能に，多い場合，特に 0D Sliding を用いた場合は，予測よりも高性能となっていた．原因については幾つか考えられるが，詳細については今後検証を進めたい．

今後，前述した故障台数増加時の性能低下を軽減する手法について検討するとともに，メッシュ/トーラス以外のネットワークトポロジや，ステンシル通信以外のパターンに対しての評価を進めていく．また，現在は実アプリケーションを用いた実験を進めており，その評価も進めていく

予定である．

謝辞 本研究は，科学技術振興機構 (JST) 戦略的創造研究推進事業 (CREST) における研究領域「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」研究課題「メニーコア混在型並列計算機用基盤ソフトウェア」によるものである．

参考文献

[1] Cappello, F., Geist, A., Gropp, W. D., Kale, S., Kramer, B. and Snir, M.: Toward Exascale Resilience: 2014 Update, *Supercomputing Frontiers and Innovations*, Vol. 1, pp. 1–28 (online), available from ⟨http://superfri.org/superfri/article/view/14/7⟩ (2014).

[2] Message Passing Interface Forum: MPI: A Message-Passing Interface Standard Version 3.1, Message Passing Interface Forum (online), available from ⟨http://www.mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf⟩ (accessed 2015-9-1).

[3] Fault Tolerance Research Hub: User Level Failure Mitigation — Fault Tolerance Research Hub, Fault Tolerance Research Hub (online), available from ⟨http://fault-tolerance.org/ulfm/⟩ (accessed 2015-9-1).

[4] Bland, W., Bouteiller, A., Herault, T., Bosilca, G. and Dongarra, J.: Post-failure recovery of MPI communication capability: Design and rationale, *International Journal of High Performance Computing Applications*, Vol. 27, No. 3, pp. 244–254 (online), DOI: 10.1177/1094342013488238 (2013).

[5] 竹房あつ子，中田秀基， 池上努，戸澤貴之，田中良夫：耐障害性ミドルウェア falanx への高可用分散協調スケジューラの実装，研究報告ハイパフォーマンスコンピューティング (HPC)，Vol. 2014, No. 8, pp. 1–8 (オンライン)，入手先 ⟨http://ci.nii.ac.jp/naid/170000084387/⟩ (2014).

[6] Teranishi, K. and Heroux, M. A.: Toward Local Failure Local Recovery Resilience Model using MPI-ULFM, *21st European MPI Users' Group Meeting, EuroMPI/ASIA '14, Kyoto, Japan - September 09 - 12, 2014*, p. 51 (online), DOI: 10.1145/2642769.2642774 (2014).

[7] 吉永一美，亀山豊久， 堀敦史， 石川裕：エクサスケールでの耐故障性実現に向けた代替ノード配置による通信性能の評価，情報処理学会研究報告. [ハイパフォーマンスコンピューティング]， Vol. 2014, No. 16, pp. 1–6 (2014).

[8] 吉永一美，亀山豊久，畑中正行， 堀敦史， 石川裕：代替ノード利用手法による耐故障性実現に向けた通信性能の評価と検討，情報処理学会研究報告. [ハイパフォーマンスコンピューティング]， Vol. 2014, No. 6, pp. 1–8 (2014).

[9] 吉永一美，亀山豊久， 堀敦史， 石川裕：予備ノードを利用した故障後の実行継続手法の検討と評価，情報処理学会研究報告. 計算機アーキテクチャ研究会報告，Vol. 2014, No. 21, pp. 1–9 (2014).

[10] 吉永一美， 堀敦史， 石川裕：ステンシル計算におけるスペアノードを用いた故障復帰における通信性能への影響について，ハイパフォーマンスコンピューティングと計算科学シンポジウム論文集，Vol. 2015, pp. 9–16 (2015).

[11] Hori, A., Yoshinaga, K., Herault, T., Bouteiller, A., Bosilca, G. and Ishikawa, Y.: Sliding Substitution of Failed Nodes, *Proceedings of the 22nd European MPI Users' Group Meeting*, EuroMPI '15, ACM (2015).

[12] 志田直之，住元真司，宇野篤也：スーパーコンピュータ「京」の MPI と低レベル通信，雑誌 Fujitsu， Vol. 63, No. 3, pp. 299–304 (2012).