Vol.2014-HPC-145 No.6
2014/7/28

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

吉永 一美[1,a)]　亀山 豊久[1]　畑中 正行[1]　堀 敦史[1]　石川 裕[2,1]

概要：莫大なハードウェアにより構成されるエクサスケール環境では，システム全体の MTBF が短縮されるため，その実現には耐故障性の確保が不可欠である．現在主流である故障対策手法は，システムレベルでのチェックポイント・リスタートである．しかし，システムの大規模化に伴う保存データサイズの増大により，故障対策の時間がアプリケーションの実行時間を圧迫してしまい，エクサスケール環境での適用は現実的ではない．そこで，故障対策をシステムに一任せずアプリケーションと連携し，ユーザレベルでの故障対策を行う Fault Resilience が提案されている．
我々はエクサスケールでの Fault Resilience 環境において，ユーザレベルでの故障対策をどのような手法で実装し，故障後の実行を継続させるべきかについて検討を進めている．現在はステンシル計算アプリケーションを対象とし，代替ノード利用手法を用いた故障からの復帰手法の評価を進めている．代替ノード利用手法では，故障したノードの代わりに予め確保していた予備ノードを用いることで実行を継続する．代替ノードを用いた場合，故障前とは異なるノード間での通信が発生することになり，通信の衝突が発生し通信性能が低下する可能性がある．本論文では，ステンシル計算アプリケーションに代替ノード利用手法を実装し，その通信性能の評価を行う．通信性能の低下の要因は，複数通信が同一通信経路を共有するために発生する通信衝突であると考え，その関係性について明らかにする．さらに，通信経路を制御することで衝突を回避し，通信性能の低下を回避する手法を提案し，その有効性を示す．

1. はじめに

エクサスケール環境においてはハードウェアの規模が莫大になるために故障率が上昇し，システム全体の MTBF(平均故障間隔/Mean Time Before Failure) が短くなることが予想される．このため，エクサスケールの実現に向け，耐故障性の確保は重要な課題となっている．

現在の HPC 環境における耐故障手法の主流は，システムレベルでのチェックポイント・リスタートを用いたものである．この手法では，システムにより定期的にアプリケーションのスナップショットをストレージへと保存し，故障発生後にそのスナップショットから状態を復元して実行を再開する．しかしエクサスケールでは，システムの大規模化に伴いスナップショットのデータ量が増加する．そのため，チェックポイント作成時のストレージへの書き込み，及びリスタート時のスナップショット読み込みに長時間を要する．その一方で，システムの MTBF はシステムの大規模化に伴い短くなるため，故障が発生しない正常な状態の大半をチェックポイント・リスタートの処理に費やす可能性がある．最悪の場合は MTBF を上回るチェックポイント作成時間となり，アプリケーションの実行が完全に停止することも考えられる．このため，エクサスケール環境においては単純なシステムレベルのチェックポイント・リスタート手法は破綻すると言われている [1].

以上のような背景から，故障への対応をシステムに一任せず，アプリケーションと連携した効率的な故障対策を実現する，Fault Resilience という考え方が注目されている．例えば，アプリケーションと連携することにより，スナップショットとしてオンメモリの全てのデータを保存するシステムレベルでのチェックポイントではなく，実行再開に必要なデータのみを保存するチェックポイント処理が実現できる．このような処理によりスナップショットの作成・読み込み時間を大幅に短縮し，エクサスケールのような大規模環境における故障対策が可能となる．

ULFM(User Level Failure Mitigation)[2], [3] は，Fault Resilience を実現するためのミドルウェアの一つである．ULFM は OpenMPI をベースに Fault Resilience を実現する機構を追加した MPI 実装であり，MPI Forum の Fault

[1] 理化学研究所 計算科学研究機構
RIKEN AICS
[2] 東京大学
The University of Tokyo
[a)] kazumi.yoshinaga@riken.jp

Tolerance Working Group[4] により設計され，テネシー大学を中心として開発が進められている．ULFM は故障として Process Failure を対象とし，実行中にノードの故障などでプロセスが落ちた場合に，その故障のプログラムへの通知や，故障したプロセスを除いた新たなコミュニケータの生成など，ユーザレベルでの故障対策を実現するための機構を実装し，利用するための API を提供している．

我々は，ULFM のようなユーザレベルでの故障対策を実現するための機構を用いた，エクサスケールでの Fault Resilience 機構の実現に向け，故障からの復帰手法について研究を進めている．ここで，チェックポイントの作成はアプリケーション内で行われ，保存されたスナップショットは全てのプロセスからアクセス可能なストレージへ保存されていることを仮定している．チェックポイントを用いた故障からの復帰には，故障ノードを除いた残りのノードを用いた実行や，故障ノードの代わりとなる代替ノードを利用した実行など，様々な手法が考えられる．そこで故障からの復帰手法について，その復帰後の性能や実装の容易性などの観点から検討し，どのような手法を用いることが適切かつ有効であるか評価を進めている．

現在はステンシル計算アプリケーションを対象とし，代替ノード利用手法を用いた故障復帰の実現について性能評価を進めている [5]．代替ノード利用手法は，故障ノードの代わりに予め確保していた代替ノードを利用することで，プログラムを大きく変更せずにロードバランスを保ったまま故障後も実行を継続する手法であり，容易に実装が可能である．しかし，ネットワーク的に離れたノード間での通信が生じるため，故障後の実行では通信遅延の増大や通信衝突の発生などにより，性能の低下を招く可能性がある．

本論文では，代替ノード利用手法を用いたステンシル計算アプリケーションの故障対策について，復帰後の通信性能を京を用いて検証し評価する．通信性能の低下の大きな要因は通信衝突であると考え，衝突を引き起こす原因となる複数通信による通信経路の共有と，全体の通信性能の関係性について明らかにする．また，通信経路を制御することにより通信性能の低下を回避する手法について，実験によりその有効性を示す．

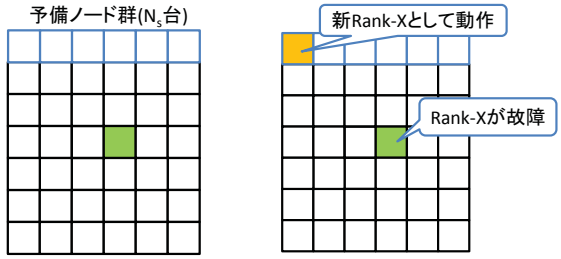
まず，第 2 章において，代替ノード利用手法によるステンシル計算に対するユーザレベルでの故障対策手法について，その特徴を述べる．次に，第 3 章において，代替ノード利用手法を用いた際の故障後の通信性能について，京を用いた実験結果に基づき論ずる．そして第 4 章にて関連研究について述べ，第 5 章でまとめを行う．

2. 代替ノード利用手法を用いたステンシル計算

本章では，代替ノード利用手法を用いたステンシル計算について説明する．ここでは，2 次元メッシュネットワーク環境上で 2 次元ステンシル計算を実行した場合を挙げて説明を行う．また、ネットワークのルーティングアルゴリズムは XY ルーティングを例として用いる．

2.1 代替ノード利用手法の適用

代替ノード利用手法を用いたステンシル計算の実行では，実際に計算するノードに加えて予備ノードを実行時に予め確保する．そして，実行中に計算に利用しているノードに故障が発生した際に，そのノードの代わりに予備ノードを利用して実行を継続する．



(a) 故障前の実行状態　(b) ノード故障後の実行継続例

図 1 代替ノード利用手法による 2 次元ステンシル計算の故障対策

図 1(a) は，6 × 6 の 2 次元ステンシル計算を代替ノード利用手法にて実行した例である．図中において青の四角で示されるノードは，実行時に確保した予備ノード群である．ここで，図中の緑の部分のノードが故障した時，**図 1(b)** のように予備ノード群の一つを代替ノードとして利用する．故障したノードが故障前に担当していた計算領域をこの代替ノードに担当させ，MPI ランクも同一に設定した上で故障前の計算を継続する．つまり，故障発生前と比較して，プロセスが実行されるノードが変更されただけとなるために，復帰後にもノード間でのロードインバランスは発生せず，アプリケーション内通信パターンも隣接ランクのプロセスとの通信のまま変わらない．そのため，代替ノード利用手法を用いた故障対策の実現では，計算や通信などのコアな部分に手を入れる必要はなく，チェックポイント・リスタート処理や利用するノードの変更といった故障対策自体のプログラムを実装するのみで，アプリケーションの耐故障性を容易に確保することができる．

例えば ULFM を用いて，ステンシル計算アプリケーションへ代替ノード利用手法を実装することは，非常に容易である．まず，計算ノードのみで構成されるコミュニケータを定義し，通常実行時にはそのコミュニケータを用いて通信を行う．そして故障を検知した場合，ULFM の機能を利用して現在利用しているコミュニケータから故障ノードを除いた新たなコミュニケータを構成し，そのコミュニケータに代替ノード上のプロセスを追加する．この時，代替ノード上のプロセスのランクを故障したノードのものと同一になるよう設定する．代替ノード上のプロセスでは，

保存したスナップショットから故障したノード上で動作していたプロセスの状態を復元する．最終的に，実行に利用するコミュニケータを新たに作成したものに置換することで，故障前と同様の動作を継続して実現できる．

なお，本手法を用いた場合，予備ノードの台数が N_s 台あれば，N_s 回の故障に耐えることが可能である．ノードの故障率を λ とすれば，N_s 回の故障が発生する確率は λ^{N_s} となり指数関数的に故障率は低下する．

2.2 代替ノード利用手法の問題点

代替ノード利用手法の問題点として，予備ノードの確保によるオーバーヘッドと，復帰後の通信性能の低下の二つが上げられる．

まず，予備ノードの確保によるオーバーヘッドについてであるが，本手法では実際に処理を行う計算ノード N 台に加え，故障発生前は何も行わない予備ノードを N_s 台確保しているため，確保した全ノードに対する計算性能は $\frac{N}{(N+N_s)}$ となる．故障発生後も計算に用いるノード数は N 台で変化しないため，故障発生前後に関わらずこの性能である．しかし，計算ノードの台数 N が十分に大きければ，予備ノードの確保によるオーバーヘッドの影響は非常に小さくなり，許容できると考えている．例えば，100×100 の 2 次元ステンシル計算において，1 行分である 100 ノードを予備ノードとして確保した場合の性能低下率は $(1 - \frac{100^2}{100^2+100}) * 100$ となり，0.99%である．同様に $100 \times 100 \times 100$ の 3 次元ステンシル計算に対し，その 1 面分である 100×100 ノードを予備ノードとして追加確保した場合，性能低下率は $(1 - \frac{100^3}{100^3+100^2}) * 100$ であり，こちらも 0.99%となる．

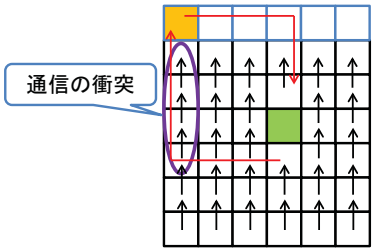


図 2 代替ノード利用で発生する他プロセスの通信との衝突

続いて，復帰後の通信性能の低下について述べる．代替ノード利用手法を利用した場合，故障発生後の実行においてもプログラム上では隣接ランクのプロセスとの通信であるが，代替ノードにプロセスを配置したことにより，ネットワークトポロジ的に離れた位置にあるノード間での通信となる．そのため，故障前と比較して通信遅延が増加する上に，プロセスの配置が変更された事で通信の衝突が発生する可能性が生じ，通信性能が低下する．

図 2 は先ほどの図 1(b) の例で実行を継続した 2 次元ステンシル計算での，上方向への通信パターンのみを示したものである．故障発生前はすべての通信が隣接ノードへの通信であり，衝突が発生することは無い．しかし，代替ノードを利用したことによる通信経路の変化により，丸で囲んだ部分において，故障する前から存在する通信との通信経路の共有が発生するため，通信の衝突が生じる可能性がある．

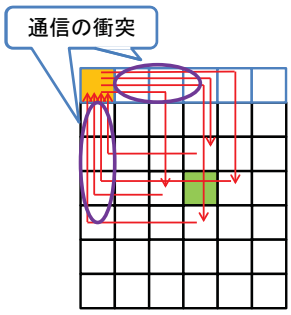


図 3 代替ノード利用で発生する自プロセスの通信の衝突

また，**図 3** は代替ノードを利用することで発生した通信の通信経路をすべて示したものである．代替ノードから隣接プロセスへの通信，または隣接プロセスから代替ノードへの通信が，丸で囲まれた部分において同一経路を通過し，最大で 4 通信が経路を共有しており，通信衝突が発生する可能性が生じる．なお，図中においては，故障ノードのネットワークはそのまま利用可能な状況で示しているが，利用不可能となり迂回した場合も通信衝突の最大数はこの例と変わらない．

図 2 で示した通常の通信との経路共有と，図 3 で示した代替ノードからの通信または代替ノードへの通信による経路共有の双方の影響により，2 次元ステンシル計算では最大で 5 通信が同一経路を共有し，衝突する可能性がある．3 次元ステンシル計算の場合を考えれば，代替ノードから隣接プロセスへの通信，隣接プロセスから代替ノードへの通信はそれぞれ 6 通信である．そのため，経路を共有する通信数は，最大で 7 通信になる可能性がある．

我々はステンシル計算に対する故障対策手法として，プログラムへの変更が少ない為にアプリケーションプログラマへの負担が少なく，復帰後にもロードバランスが保たれる代替ノード利用手法を採用することを考えている．そこで，代替ノード利用手法による通信性能の低下について評価と検討を行うために，京を用いて実験を行った．次章ではその実験について述べる．

3. 京を用いた代替ノード利用手法の通信性能実験

本実験では，ステンシルパターンの通信のみを行う実験用プログラムを作成し，代替ノードを利用した場合の，故障の前後での通信性能の変化について調査を行った．故障

するノードは 1 台のみと設定し，復帰後もネットワークは継続して利用可能な状態であると仮定している．

実験環境として，ネットワークトポロジが 6 次元メッシュ/トーラスである京を用いた．京では，投入ジョブの形状指定は 1 次元・2 次元・3 次元いずれかの仮想形状で行うことができる．投入されたジョブは指定された仮想形状を形成するように tofu ネットワーク上の 6 次元座標へとマッピングされ，仮想形状での隣接ノードは，マッピングされた tofu 6 次元空間においても隣接した 1 ホップの通信となることが保証される [6]．そのため故障発生前の状態では，全てのステンシル通信は衝突せずに 1 ホップで完了する．

3.1 実験概要

実験に用いたプログラム内の通信には，通常の MPI による通信ではなく，京の低レベル通信ライブラリである tofu ライブラリ [7] を用いた RDMA 通信機能を利用している．低レベル通信ライブラリを用いた理由は以下の 3 点である．

a) 送受信に利用する RDMA エンジンを指定可能
京は RDMA エンジンを 4 本搭載しているが，通常の MPI 通信では送信元および受信先の RDMA エンジンの指定ができない．そのため，RDMA エンジンが受け持つ通信数に偏りが生じ，通信性能が低下するおそれがある．この要素を排除し，ステンシルの通信方向毎に利用するエンジンを固定することで，代替ノードを利用した場合の通信性能自体の詳細な解析を可能とする．

b) 通信経路制御の実現
今回の実験では，通常経路を用いた通信実験の他に，通信経路の制御による代替ノード利用手法の通信性能向上実験を行う．MPI 通信では通常の経路しか利用できないが，tofu ライブラリによる低レベル通信を行うことで，この経路制御が実現できる．

c) MPI 処理による通信オーバーヘッドの除去
MPI 通信処理による通信オーバーヘッドを取り除くことで，代替ノード利用による通信性能への影響自体の解析を詳細に行う．

京で提供される MPI ライブラリは，拡張機能を用いることで a) 及び c) を実現できるが，b) の通信経路制御は行えないため，tofu ライブラリを用いた通信を実装し，利用した．

通信性能の測定方法は以下の通りである．まず，故障発生前の状態 (通常時) において計算ノードを用いた通信を実行し，プロセス毎に通信時間を計測する．プログラム内ではステンシル通信の直前にバリア同期を行っているため，各プロセスで測定された通信時間のうち，最大のものが全体の通信に要した時間となる．この値を通常時の通信時間とみなす．その後，計算ノードのうち一つを故障したとみなし，そのノードの代わりに予備ノードの一つを代替ノードとして利用した通信時間の測定を行う．ここでも同様にプロセス毎に計測を行い，最大のものを全体の通信完了時間とみなす．この測定を計算ノードと予備ノードの全組み合わせについて実行し，代替ノード利用による通信性能の変化を調査した．

3.2 2 次元ステンシル通信を用いた代替ノード利用手法の性能評価

まず，2 次元ステンシル通信プログラムを用いた性能評価を行った．2 次元ステンシル通信プログラムでは，上下左右の 4 方向の通信に対して，それぞれに別の RDMA エンジンを指定し，通信間での RDMA エンジンの競合を排除している．

この実験では，2 次元形状指定で 6 × 12(1x1x6x2x3x2) のジョブを投入する．括弧内は実行時にマッピングされた tofu 6 次元空間である．確保された 6 × 12 ノードのうち，$y = 11$ である 6 ノードを予備ノード群として確保し，残りの 6 × 11 ノードを用いてステンシル通信を行った．通信データサイズは 64KiB・256KiB・1MiB・4MiB の 4 種類で測定を行った．

3.2.1 単一方向通信のみでの通信性能

通信衝突による性能の変化を調査するために，まずは 2 次元ステンシル通信のうち右方向の通信のみを行い，代替ノード利用時の性能について調査した．

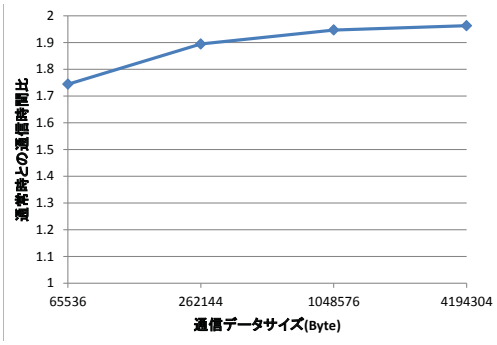


図 4 代替ノードを用いた単一方向通信の通信性能

図 4 に計測結果を示す．横軸は 1 通信あたりのデータサイズであり，縦軸は通常時の通信時間を 1 とした場合の比率として，故障ノードと代替ノードの全組み合わせにおける通信時間の中の最大値を示している．代替ノードを用いた通信では，通常時と比較して最大の通信時間は約 2 倍となっていることが確認できる．

この実験では単一方向のみに通信を行っているため，各プロセスでの通信は送信と受信がそれぞれ一つずつである．そのため，図 3 に示されるような代替ノード上のプロセスから発生する通信間での経路の共有，及びそのプロセスへと送られる通信間での経路の共有は発生せず，図 2 で示した通常時から存在する通信との経路を共有のみが発生する．つまり，同一経路上には最大でも二つの通信が存在す

るだけである．二つの通信により経路が共有されると，通信の衝突が発生する可能性がある．衝突が発生した場合，その経路において 1 通信時間分の通信待ち時間が発生し，通信時間は 2 倍になる．二つの通信が経路を共有する経路は複数存在するが，衝突による通信待ち時間の間に衝突しないその他の通信は全て完了するため，衝突回数は 1 回のみである．そのため，全体の通信完了に要する時間は，通信衝突による待ち時間の 1 通信時間分が増加した，通常時の 2 倍になると考えられる．実際の通信においてはデータはパケットに分割されるため，二つのデータが衝突する際には 2 データのパケットが混在して通信されるが，最終パケットの到着を通信完了と考えれば，内部でのパケット通信順序とは無関係に全体の通信時間は 2 倍である．

3.2.2 2 次元ステンシル通信の通信性能

次に，2 次元ステンシル計算の通信を行った場合の通信性能について調査した．**図 5** に実験結果を示す．このグラフも同様に横軸は通信データサイズ，縦軸は通常時を 1 とした場合の通信時間比である．実験の結果，代替ノードを用いた実行により，通信時間は通常時と比較して約 2.9 倍に増加することが確認された．

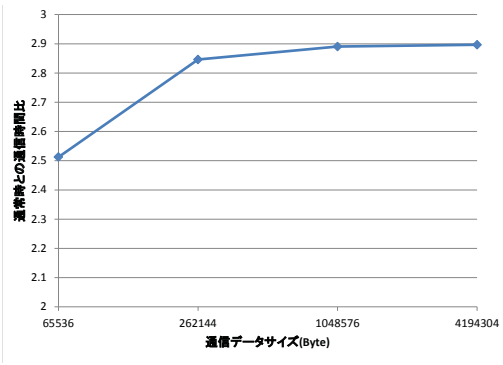


図 5 代替ノードを用いた 2 次元ステンシル通信の通信性能

第 2 章で述べたように，2 次元ステンシル計算に代替ノード利用手法を適用した場合，故障発生後の実行では最大で 5 通信が同一経路上に存在する可能性がある．そこで，故障ノードと代替ノードの組み合わせ毎に各通信が利用する通信経路を求め，経路を共有する最大の通信数をカウントし確認した．この確認は，実験で用いた各ノードの物理 6 次元座標をログとして記録し，それを解析することで行った．京のネットワークである tofu ネットワークは，ルーティングポリシーとして拡張次元オーダールーティングを採用しており．通信パケットは (B，C，A，)X，Y，Z，A，C，B の順に 6 次元座標軸上を推移する．括弧内の最初の ABC 軸ルーティングはネットワーク故障時に迂回経路を利用する場合等に使われるもので，実際にハードウェア故障が発生していない今回の実験時には，X，Y，Z，A，C，B の順にルーティングされる [6]．このルーティングアルゴリズムに基づき，送信元ノードの座標と受信先ノードの座標から全ての通信経路を辿ることで，各経路を利用する通信を求めることが可能である．上記の方法を用いて確認した結果，本実験において経路を共有する最大通信数は 5 となっていた．

5 通信により通信経路が共有された場合，最大で五つの通信が衝突し，その結果通信に要する時間は単一方向通信の 5 倍になると考えられる．そこで，3.2.1 小節で計測した通常時の単一方向通信時間と，本実験の結果である故障時の 2 次元ステンシル通信時間との比率を求めた．その結果を**図 6** に示す．グラフに示される通り，代替ノードを用いた 2 次元ステンシル通信の最大通信時間は，通常時の単一方向通信時間の約 5 倍となっていることがわかる．

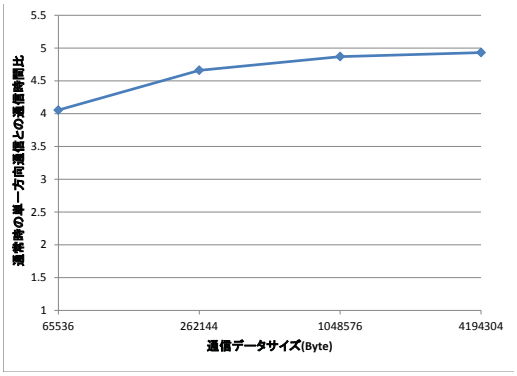


図 6 通常時の単一方向通信時間を基準とした代替ノード利用時の 2 次元ステンシル通信時間比

しかし本実験において，2 次元ステンシル通信の代替ノード利用時における最大通信時間は，通常時と比較して約 2.9 倍となっており，5 倍ではない．この理由は，4 方向に通信を行う 2 次元ステンシル通信が，通常時において単一方向通信よりも通信時間を要しているからである．京の tofu インターコネクトは RDMA エンジンを 4 つ備えており，4 通信を同時に実行可能である．今回のプログラムでは 4 通信それぞれに個別の RDMA エンジンを割り当てており，理想的には 2 次元ステンシル通信と単一方向通信は同時間で完了するはずである．しかし実際には，メモリと RDMA エンジン間を結ぶインターフェースの帯域幅による制限から，4 通信時には通信性能が低下する [7]．そのために，2 次元ステンシル通信の完了には単一方向通信時間と比較して，**図 7** に示すように約 1.7 倍の時間を要していた．その結果，代替ノードを用いた 2 次元ステンシル通信時間は，通常時の 2 次元ステンシル通信時間と比較すれば最大で $5/1.7 = 2.9$ 倍となる．

以上から，代替ノード利用手法による 2 次元ステンシル通信の故障発生後の通信時間 t_{alt} は通常時の単一方向通信時間を t_{single}，代替ノード利用時に経路を共有する最大通信数を C_{max} とすれば，

$$t_{alt} = t_{single} \times C_{max} \quad (1)$$

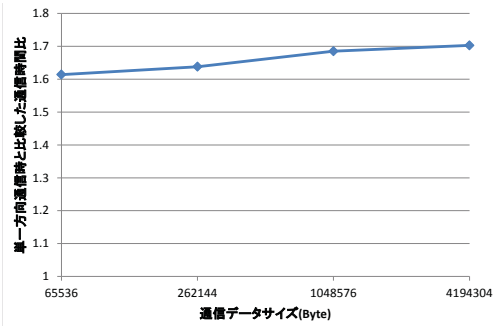


図 7 通常時における単一方向通信と 2 次元ステンシル通信の通信時間比

と表すことができる．つまり，代替ノード利用手法を利用した場合，故障前を基準とした復帰後の 2 次元ステンシル通信時間の比 r_{alt} は，

$$r_{alt} = \frac{t_{single} \times C_{max}}{t_{stencil}} = \frac{C_{max}}{r_{stencil}} \quad (2)$$

となる．ここで，$t_{stencil}$ は通常時の 2 次元ステンシル通信時間，$r_{stencil}$ は．通常時の単一方向通信時間を基準とした通常時の 2 次元ステンシル通信時間の比である．つまり，代替ノード利用手法を用いた場合，復帰後の通信時間は経路を共有する通信の最大数により決定し，これをを減少させることで通信時間を短縮できることを示している．

図 8 は，式 2 を確認するために，データサイズ 4MiB での 2 次元ステンシル通信の実験結果を，経路を共有する最大通信数ごとに分類し，各最大通信数の最大通信時間を用いて性能を示したものである．青線で示されるグラフが実測値であり，緑線が式 2 により求めた通信性能の予測値である．経路を共有する通信の数が 2，4，5 の場合はほぼ実測値と予測値が同等の値になっているが，3 の場合には実測値と予測値に開きがある．この原因については現在調査中である．

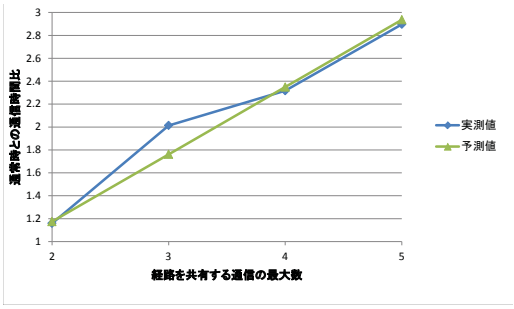


図 8 通信経路共有数ごとの通信時間比較

3.3 通信経路制御による代替ノード利用手法の性能向上

3.2.2 において，代替ノード利用時の通信性能は，経路上に存在する通信の最大数と線形関係にあることを示した．2.2 で述べたように，代替ノード利用時に発生するステンシル通信における経路の共有の多くは，代替ノードからの通信と，代替ノードへの通信が原因である．そこでこれらの通信に着目し，それぞれの通信の経路を変更することによって通信経路の共有数を減少させ，全体の通信性能の向上を試みた．今回の実験環境である京では，低レベル通信 API を利用することにより，拡張次元オーダールーティングの最初の ABC 軸の移動を制御可能である．この機能を用いて実際に代替ノードからの通信と代替ノードへの通信に対して経路制御を実装し，2 次元ステンシル通信を用いた実験を行った．

代替ノードから隣接プロセスに対しての通信について，最初にルーティングされる ABC 座標を以下の表 1 の通り設定する．表中の a，b，c はそれぞれ代替ノード自身の A，B，C 座標である．

表 1 代替ノードからのステンシル通信に対する経路制御

ステンシル通信方向	ルーティング経路		
	A 座標	B 座標	C 座標
右方向	a	b	c
左方向	(a+1) mod 2	b	c
上方向	a	(b+1) mod 3	c
下方向	a	b	(c+1) mod 2

隣接プロセスから代替ノードへの通信については，代替ノードからその隣接プロセスへの通信の逆経路となるように経路を設定した．例えば，代替ノードの左側プロセスからの通信は，(A，B，C) を ((a+1) mod 2，b，c) として設定する．

図 9 に，経路制御の実装の有無による代替ノード利用手法の通信性能を示す．経路制御を実装することで，通常時と比較した通信時間の増加を，最大 2.9 倍から最大 1.75 倍にまで軽減できており，経路制御が有効であることを示している．

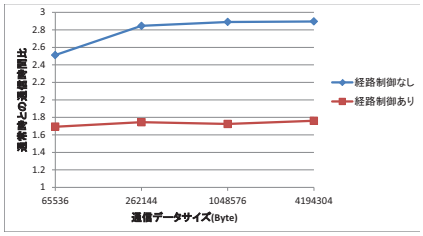


図 9 経路制御機構を実装した代替ノード利用手法による 2 次元ステンシル通信性能

3.4 3 次元ステンシル通信の通信性能

続いて，代替ノード利用手法による 3 次元ステンシル通信の通信性能について調査した．3 次元ステンシル通信では同時に 6 方向の通信を行う．京の RDMA エンジンは 4 本であるため，使用する RDMA エンジンの競合を完全に排除することは不可能である．今回実装したプログラムで

は，まず XY 方向の 4 通信をそれぞれ 1 から 4 の RDMA エンジンに割り当て，残りの Z 方向の 2 通信をそれぞれ RDMA エンジン 1 と 2 に割り当てて通信を行っている．

3 次元ステンシル通信実験では，3 次元形状指定で 6 × 6 × 6(3x2x3x2x3x2) のジョブを投入し，$z = 5$ である 6 × 6 ノードを予備ノード群とし，残りの 6 × 6 × 5 ノードで 3 次元ステンシル通信を行う．通信データサイズは 2 次元ステンシル通信実験と同様に，64KiB・256KiB・1MiB・4MiB の 4 種類を設定した．

図 10 はその実験結果である．縦軸は通常時の 3 次元ステンシル通信時間を基準とした比率を示している．代替ノードを用いた実行により，復帰後の通信時間は通常時と比較して約 2.1 倍に増加していることがわかる．

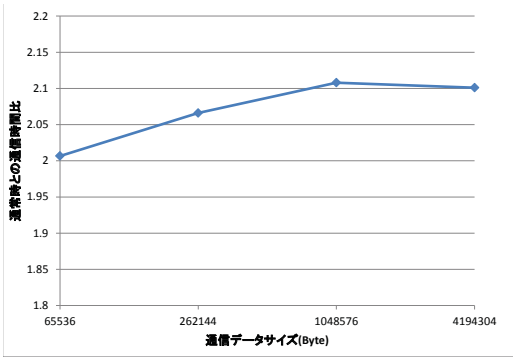


図 10 代替ノードを用いた 3 次元ステンシル通信の通信性能

3 次元ステンシル通信では各プロセスが 6 通信を行うため，代替ノード利用時に経路を共有する通信数は最大で 7 である．本実験で投入したジョブが用いたノード配置から実際に経路求めて確認したところ，今回のジョブでの最大共有数は 6 であった．また，通常時の単一方向通信時間と比較して，通常時の 3 次元ステンシル通信時間はおよそ 3.7 倍になっていた．2 次元ステンシル通信の場合と同様に式 2 を用いて考えた場合，復帰後の通信時間は通常時の約 1.6 倍と予測されるが，実験の結果はこれを大きく上回っている，

式 2 においては，3.2.1 で述べた「一度衝突した通信は，その後は別のノード間の隣接通信とは衝突しない」という前提に基づき，経路を共有する通信数を衝突回数とみなした形で通信時間の増加を求めている．3 次元ステンシル通信では同時に 6 方向の通信を発行するが，実験環境である京の RDMA エンジンの本数は 4 本であり，残り 2 方向の通信はその前に発行された通信が完了した後に開始する．そのため，最初に発行された通信が一度衝突した後も別の隣接通信と衝突する可能性があり，この前提が崩れてしまう．つまり，式 2 で示される性能は通信数が RDMA エンジン数以下である場合にのみ有効であり，今回の 3 次元ステンシル通信の実験のように通信数が RDMA エンジン数を上回る場合は，それを考慮した別のモデルが必要であると考えている．

4. 関連研究

本研究と同じく，ユーザレベルでの故障復帰手法に関する研究として，Ali らによる [8] がある．この論文で提案されている故障復帰手法では，実行時に故障が発生した場合，故障した MPI プロセスの代わりとなる MPI プロセスを新たに作成 (spawn) することで，故障前と同一のプロセス数を維持し，復帰後のロードバランスを保っている．そして実際にこの手法を，偏微分方程式を解くアプリケーションに対して ULFM を用いて実装し，故障からの復帰に必要な時間についての評価を行っている．

代替プロセスを用いてロードバランスを保つと言う点では，我々の提案する代替ノード利用手法と同様であるが，新たに作成される MPI プロセスのノード配置については言及がなく，復帰後の通信性能に関する議論もされていない．

5. まとめ

本論文では，エクサスケールでの Fault Resilience 環境の実現に向け，代替ノード利用方式によるステンシル通信に対するユーザレベルでの故障対策手法について，その通信性能の評価を行った．

代替ノードを利用した実行によって通信経路が変更され，複数の通信による経路の共有が発生することにより，通信衝突の可能性が発生し通信性能が低下する．実際に京を用い，代替ノードを用いた 2 次元ステンシル通信による測定を行った結果，その通信時間は経路を共有する通信数に線形に依存することを確認した．そこで，通信性能を改善する方法として，代替ノードに関係する通信の経路制御を行い経路を共有する通信数を減少させる手法を提案した．今回の実験では，経路制御を行わない場合は通信時間が最大で 2.9 倍となっていたが，経路制御により最大で 1.75 倍に抑えられ，その有効性を示した．また，3 次元ステンシル通信プログラムによる実験を行った結果，実際の通信時間が予測値よりも悪化することを確認した．これは，各ノードの RDMA エンジンの本数を発行する通信数が上回るために，経路を共有する通信数と実際の通信衝突回数が同一であるという仮定が崩れたためである．このような場合の性能については，今後更に検証を進めていく．

その他の今後の課題として，京とは異なる別環境での通信性能の評価が挙げられる．ネットワークトポロジが変化すると，衝突が発生する状況や隣接プロセス通信時間の不均一など，条件が大きく変わることが予想されるため，代替ノード利用手法の通信性能について検証を進める必要がある．また，ステンシル以外の通信パターンについても代替ノード利用手法を適用し，その通信性能について検討を行う予定である．そしてこれらの結果により得られた知見

から，故障復帰後にも高性能な性能を維持する手法の実現を目指す．

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, B., Kale, L., Kramer, B. and Snir, M.: Toward Exascale Resilience, *Int. J. High Perform. Comput. Appl.*, Vol. 23, No. 4, pp. 374–388 (2009).
[2] Fault Tolerance Research in the Innovative Computing Lab at the University of Tennessee: User Level Failure Mitigation — ICL Fault Tolerance, The University of Tennessee (online), available from ⟨http://fault-tolerance.org/ulfm/⟩ (accessed 2014-4-30).
[3] Bland, W., Bouteiller, A., Herault, T., Hursey, J., Bosilca, G. and Dongarra, J.: An evaluation of User-Level Failure Mitigation support in MPI, *Computing*, Vol. 95, No. 12, pp. 1171–1184 (online), DOI: 10.1007/s00607-013-0331-3 (2013).
[4] MPI Forum: MPI Forum's Fault Tolerance Working Group, MPI Forum (online), available from ⟨https://svn.mpi-forum.org/trac/mpi-forum-web/wiki/FaultToleranceWikiPage⟩ (accessed 2014-6-30).
[5] 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).
[6] Yuuichirou Ajima, Tomohiro Inoue, S. H. and Shimizu, T.: Tofu: Interconnect for the K computer, *Fujitsu scientific & technical journal*, Vol. 48, No. 3, pp. 280–285 (2012).
[7] 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).
[8] Ali, M. M., Southern, J., Strazdins, P. E. and Harding, B.: Application Level Fault Recovery: Using Fault-Tolerant Open MPI in a PDE Solver, *IEEE 28th International Parallel & Distributed Processing Symposium Workshops (IPDPSW 2014)*, Phoenix, USA, pp. 1169–1178 (2014).