Vol.2014-HPC-144 No.16
2014/5/27

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

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

概要：ハードウェア規模の増加するエクサスケール環境においては MTBF の短縮が懸念されており，耐故障性の確保が重要な課題となっている．しかし，現在故障対策の主流となっているシステムレベルでのチェックポイント・リスタート手法は，大規模化に伴うチェックポインティング時のスナップショットサイズの増大により，チェックポインティングに要する時間が増大することが予想される．更に，MTBF の短縮に対応するために，チェックポイント・リスタートの頻度を増やす必要がある．その結果，故障対策の時間が実行時間の多くを占め，アプリケーションの実行が進まない状態が発生してしまう．
この問題を解決するために，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 が提案された．我々はエクサスケールでの Fault Resilience 環境の実現に向け，ユーザレベルでの故障対策をどのように実装し，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．
本論文では，2 次元ステンシル計算を対象とし，ユーザレベルでの故障対策手法として代替ノード利用手法を提案する．代替ノード利用手法では，利用する代替ノードの配置により通信性能が変化するため，その性能について評価を行った．

1. はじめに

エクサスケール環境ではハードウェア規模が増加す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．このため，エクサスケールの実現に向けて，耐故障性の確保は重要な課題である．

現在の HPC 環境において，主流となっている耐故障手法は，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，故障発生時はそのスナップショットを用いて実行を再開することで，故障対応を行う．

しかしながら，エクサスケール環境においては，単純なシステムレベルでのチェックポイント・リスタートによる耐故障手法は破綻すると言われている [1]．エクサスケールにおいてはシステムが大規模化するため，スナップショットの作成に必要となるストレージへの書き込みデータ量が大幅に増加し，チェックポインティングの時間に長時間を要することになる．その一方でシステムの MTBF は前述の通り短くなるため，チェックポインティングの時間が MTBF を上回る可能性がある．このような事態が発生すると，チェックポイント・リスタートを用いた耐故障手法では，アプリケーションの実行が全く進まなくなってしまう．

そこで，故障対応をシステムへ一任するのではなく，アプリケーションと連携した効率的な対応を実現する，Fault Resilience という考え方が注目されている．アプリケーションと連携することにより，システムレベルでのチェックポインティングのようにオンメモリの全てのデータをストレージへ書き出すのではなく，アプリケーションの復帰に必要なデータのみをアプリケーションプログラマが指定して書き出すことが可能となるために，チェックポインティングに必要な時間を大きく短縮することが可能となる．

Fault Resilience を実現するためのソフトウェアとして，テネシー大学を中心として開発が進められている ULFM(User Level Failure Mitigation)[2], [3] が存在する．ULFM は，OpenMPI をベースに Fault Resilience を実現する機構を追加した MPI 実装である．ULFM では故障として Process Failure を対象としており，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．

[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

ⓒ 2014 Information Processing Society of Japan

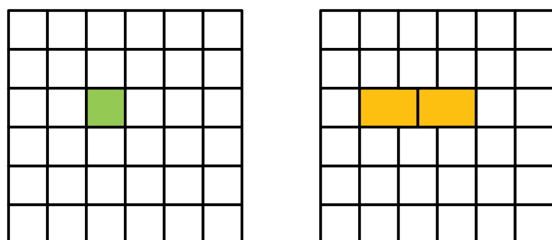
現在我々は，エクサスケールでの Fault Resilience 機構を実現するために，この ULFM を利用した耐故障手法の研究を進めている．本研究においては，チェックポインティングはアプリケーション内で行われ，保存されたスナップショットは全てのプロセスからアクセス可能なストレージへ保存されていると仮定している．本論文ではその仮定のもと，ULFM の機能を用いて故障発生後も実行を継続する際にどのような手法を用いることが適切かつ有効かを評価する．アプリケーションとして 2 次元ステンシル計算を対象とし，ULFM を用いた故障対策手法についてその性能の議論を行う．性能検証環境としてはネットワークトポロジが 6 次元メッシュ/トーラスである京を用いた．メッシュネットワークは構成に必要なハードウェア資源量がノード数 N に対して $O(N)$ と少なく，エクサスケール環境においても同様のネットワークトポロジが採用されると考えられるため，評価環境として採用した．

まず，第 2 章において，2 次元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，「しわ寄せ法」と「代替ノード利用手法」の二種類を挙げ，それぞれの特徴について述べる．次に，第 3 章において，代替ノード利用手法による故障対策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. 2 次元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

2.1 しわ寄せ法

本論文では，ノード故障が発生した際に，故障していないノードのみを用いて計算を続行する手法を「しわ寄せ法」と呼ぶ．**図 1(a)** は，2 次元メッシュネットワーク上に配置されたノード上での，6 × 6 の 2 次元ステンシル計算の例である．各四角はノードを示しており，各ノード上で一つの MPI プロセスが動作している．この図において，緑部分のノードが故障した場合，その計算領域を残りのノードに負担させ計算を続行する．**図 1(b)** はその一例であり，両横の 2 ノードが故障ノードの担当部分の計算領域を分担した場合である．



(a) 故障前の 2 次元ステンシル計算　(b) ノード故障後の実行継続例

図 1　しわ寄せ法による 2 次元ステンシル計算の故障対策

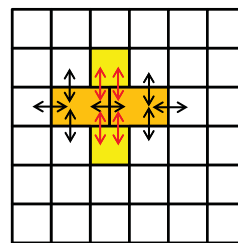


図 2　しわ寄せ法を利用した場合の通信パターンの変化例

全体の計算性能は，全体のノード数を N，故障ノード数を n とすると，$\frac{N-n}{N}$ となる．

本手法を用いた故障後の実行において問題となるのは，ノード間のロードインバランスとアプリケーション内通信パターンの変化の 2 点である．図 1(b) の例では，故障ノードが担当していた計算領域を肩代わりした両横のプロセスの計算量が，その他のプロセスの 1.5 倍となり，ロードインバランスが発生する．また，この状態で実行を再開する場合，**図 2** のようにアプリケーション内での通信パターンに変化が生じる．故障前の 2 次元ステンシル計算の通信パターンは隣接する 4 プロセスとの隣接通信であるが，しわ寄せ法を利用した実行再開後は図中の赤矢印で示される新たな通信パターンが発生する．この例では，オレンジで示した計算ノードでは 6 プロセスとの通信が，黄色で示した計算ノードでは 5 プロセスとの通信が必要となり，当初のステンシル計算の通信パターンとはかけ離れてしまう．さらに故障台数が増加すれば，より複雑なパターンが発生する．

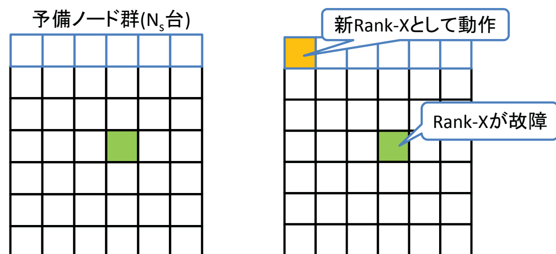
ロードインバランスの発生とアプリケーション内通信パターンの変化により，しわ寄せ法を用いた故障対策の実装では，アプリケーションプログラマが計算や通信などのコアな部分にも手を加えなければならない．プログラマの負担を軽減した耐故障手法の実装のためには，これら二つの問題点を解決する手法が必要である．

2.2 代替ノード利用手法

この手法では，アプリケーション実行時に予め余分に予備ノードを確保し，計算に利用しているノードに故障が発生した際に，故障したノードの代わりに確保していた予備ノードを利用して実行を継続する．

図 3(a) は，6 × 6 の 2 次元ステンシル計算を代替ノードを利用して実行する例である．図中の青の四角のノードは，予め確保された予備ノード群を示している．図中の緑の部分を担当していたノードが故障した時，**図 3(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し，故障ノードでの処理をこの代替ノードに担当させることで，故障後も継続した処理の実行が可能となる．つまり，プロセスが実行されるノードを変更するだけとなり，2.1 節において述べた，しわ寄せ法による実行での問題点である，ノー

ⓒ 2014 Information Processing Society of Japan



(a) 故障前の実行状態　(b) ノード故障後の実行継続例

図 3　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

ド間のロードインバランス及びアプリケーション内通信パターンの変更は発生しない．

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．代替ノードにおいては，保存したスナップショットから状態を復元する．最終的に，通信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．

本手法における全体の計算性能は，計算ノード N 台に加えて予備ノード群を N_s 台確保しているため，$\frac{N}{(N+N_s)}$ となる．この計算性能は，N_s 台のノードが故障するまで保たれる．ここで 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る．例えば，100×100 の二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追加確保した場合，性能低下率は $(1 - \frac{100^2}{100^2+100}) * 100$ となり，0.99%である．3 次元ステンシル計算に拡張し $100 \times 100 \times 100$ の計算に適用すると，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の性能低下率は，$(1 - \frac{100^3}{100^3+100^2}) * 100$ であり，こちらも同様に 0.99%となる．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．

しかし，代替ノードを利用して実行を継続した場合，プロセスの配置されるノードが変化するために，通信性能に変化が生じる．プログラム上では隣接通信であるが，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ために通信遅延が増加する．さらに，プロセスの配置が変更される事で，通信の衝突が発生する可能性がある．**図 4** は先ほどの図 3(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．ここでは，ルーティングアルゴリズムとして XY ルーティングの場合を示している．故障発生前はすべての通信が衝突の発生しない隣接ノードへの通信だったが，代替ノードを利用したことにより通信経路に変化が生じ，丸で囲んだ部分において経路の共有が発生し，通信の衝突が生じる可能性がある．

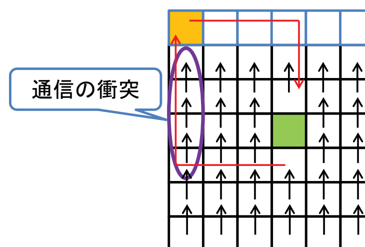


図 4　代替ノード利用で発生する他プロセスの通信との衝突

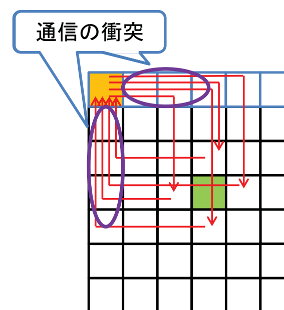


図 5　代替ノード利用で発生する自プロセスの通信の衝突

また，**図 5** は代替ノードを利用することで変化する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ており，最大で 4 通信が経路を共有する．さらに，図 4 で示した通常の通信との経路共有も発生すれば，最大で 5 つの通信が同一経路を共有し，衝突する可能性がある．

我々は 2 次元ステンシル計算に対する故障対策手法として，アプリケーションプログラマへの負担が小さく，ロードバランスも容易に保つことのできる代替ノード利用手法を採用することを考えている．その上で，生じる通信衝突による通信性能への影響を調査するため，京を用いて実験を行った．今回は故障するノードは 1 台のみと設定し，故障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．

3. 京での通信性能比較実験

本実験では，2 次元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．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/

ⓒ 2014 Information Processing Society of Japan

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ワンホップ通信となることが保証される [4].

今回の実験では 2 次元形状指定で 6 × 12(1x1x6x2x3x2) および 12 × 12(2x1x6x2x3x2) のジョブを投入した．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．6 × 12 ジョブではそのうちの 6 × 11 ノードを計算ノード，残りの 6 ノードを予備ノード群に，12 × 12 ジョブでは 12 × 11 ノードを計算ノード，残り 12 ノードを予備ノード群として，2 次元ステンシルパターンの通信を行った．通信データサイズは，6 × 12 ジョブでは 256KiB・1MiB・4MiB・16MiB，12 × 12 ジョブでは 256KiB・1MiB・4MiB と設定し，その通信時間の測定を行った．

3.1 通信性能の測定方法

まず，故障が発生していない状態 (通常時) において通信を実行し，各プロセスの通信時間を計測する．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ため，その値を通常時の通信時間とする．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 単一方向通信のみでの通信時間

通信衝突による性能の変化を調査するために，まずは四方向存在する通信を分解し，単一方向での通信のみを行い，性能の測定を行った．

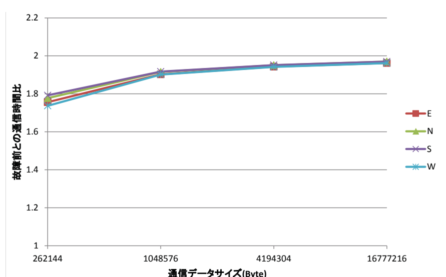


図 6 代替ノードを用いた単一方向通信の通信性能 (6 × 12 ジョブ)

図 6 と**図 7** に 6 × 12 と 12 × 12 それぞれのジョブによる測定結果を示す．凡例の E，N，S，W はそれぞれ，右，上，下，左方向の通信と対応しており，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，計測した全ての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場合，どの方向の通信においても通常時と比較して通信時間が約 2 倍となっている．

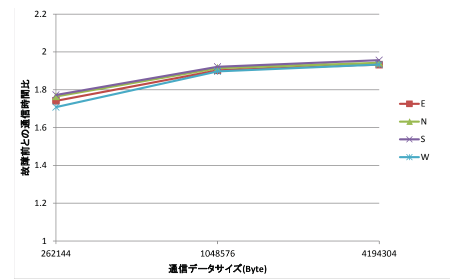


図 7 代替ノードを用いた単一方向通信の通信性能 (12 × 12 ジョブ)

代替ノードを用いたことで通信経路が変化するため，図 4 で示したように，他の通信との経路の衝突が発生する．1 方向のみの通信の場合，図 5 で述べた自プロセスでの通信の衝突は発生しないため，同一経路上に存在する通信は最大で 2 つである．通信が衝突すると，経路上で 1 通信時間分の通信待ちが発生する．衝突しなかった通信はこの通信待ち時間の間にすべて終了するため，一度通信が衝突するとその後通信は発生しない．そのため，通信衝突による通信待ち時間である 1 通信時間分が通信時間に加算され，全体の通信時間が 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，2 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．

京のネットワークルーティングポリシーは拡張次元オーダールーティングであり，パケットは B，C，A，X，Y，Z，A，C，B の順で 6 次元座標軸を推移する．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に使われるもので，基本的には X，Y，Z，A，C，B の順にルーティングされる [4]．このため，各ノードの物理 6 次元座標がわかれば，送信元座標および受信先座標から通信経路をたどることが可能である．そこで，図 6 と図 7 の実験結果について，実際に通信した経路を確認し，各プロセスの通信時間との照合を行った．その結果，通信時間が増加したプロセスにおいては，通信衝突の可能性がある経路を利用していることを確認した．

3.3 全方向通信での通信時間

次に，2 次元ステンシル計算の通信を 4 方向とも行った場合の通信性能の変化を調査した．**図 8** に実際の実験結果を示す．

2.2 節で述べたように，代替ノードを利用した場合は最大で 5 つの通信が同一経路上に存在する可能性がある．3.2

Vol.2014-HPC-144 No.16
2014/5/27

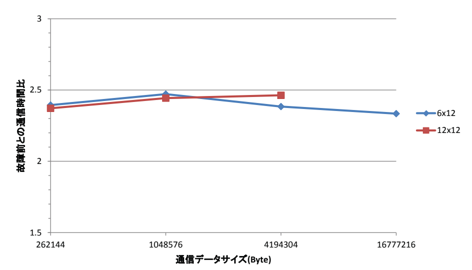


図 8 代替ノードを用いた 2 次元ステンシル通信の通信性能

節と同様に通信経路を求め，経路上に存在する最大通信数をカウントしたところ，どちらのジョブも 5 となった．そのため，通信時間は衝突の影響により最大 5 倍になると予測できる．しかし，実験では代替ノード利用時の最大通信時間は 5 倍にはならず，2.3〜2.5 倍程度にとどまっている．

この理由を考えるために，ここでは通常時の通信時間に着目する．京のインターコネクトである tofu は，4 つの通信用 DMA エンジンを持っている．そのため，2 次元ステンシル通信において 4 方向の通信を行う場合，全ての方向の通信を並列に発行できる．また，2 次元ステンシル通信は全て隣接プロセスとの通信であり，通常時は衝突なく 1 ホップの通信で全通信が完了する．そのため，全ての通信が同時に発行できる理想的な環境を想定すれば，通常時における 4 方向通信での通信時間は，単一方向通信での通信時間から変化しないはずである．

しかし，実際に通常時において測定した結果では，4 方向通信の通信時間は，単一方向通信の通信時間と比較して増加していた．**図 9** はその比較結果である．横軸は通信データサイズを，縦軸は通常時の単一方向通信での通信時間を 1 とした場合の，4 方向通信時間の比率を示している．例えば 6 × 12 ジョブ・1MiB の通信では，4 方向通信に要した時間は単一方向通信時間のおよそ 2.5 倍となっていた．

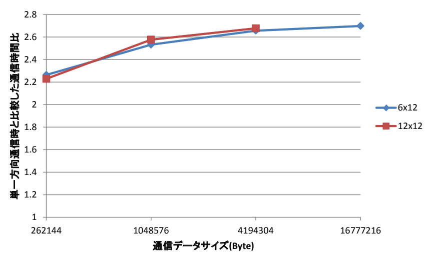


図 9 通常時における単一方向通信と 4 方向通信の実行時間比

この結果より，通常時の 4 方向通信時には，tofu の 4 つの DMA エンジンが全て同時に動作するのではなく，それぞれの通信開始時間にずれが生じていると予想される．ずれが生じた結果，4 方向通信が全て完了するまでの時間が，1 方向のみの通信が完了するまでの時間の 2.5 倍になっていると考えられる．このずれの要因としては，DMA 起動処理の発行時間や，メモリから tofu DMA エンジンへデータ転送を行う際のメモリバンド幅による制限などが考えられる．

ここで，代替ノードを用いて実行した場合の通信衝突が最大になるケースを考える．つまり，代替ノードからは途中まで経路を共有する四つの通信が発行され，さらにその共有経路上に別のステンシル通信が存在する状態である．この状態での通信を**図 10** に示す．まず一つ目のデータは，経路上に存在する別のステンシル通信と衝突し，通信完了までに 2 倍の時間を要する．二つ目以降のデータは，その前に通信を開始したデータとの経路の衝突が発生するため，最終的な全体の通信完了は，一つのデータの通信時間の 5 倍となる．

つまり，図 10 に基づけば，故障発生時に代替ノードを利用した場合の 4 方向通信時間の最大値は，通常時の 4 方向通信時間と比較して 5/2.5 = 2 倍になると考えられる．しかし実測値では 2.44 倍であり，予測される時間を上回る結果となっている．この原因の調査については今後の課題である．

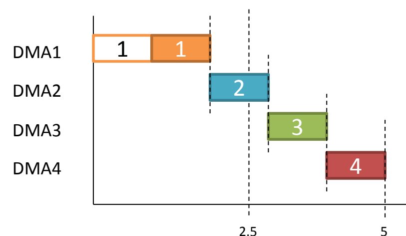


図 10 代替ノード利用時の 2 次元ステンシル通信時間

図 11 は，4 方向通信を行う 6 × 12 ジョブを用いた実験において，経路を共有する最大通信数ごとに分類し，その最大通信時間を示したものである．この結果より，衝突数が少ないと通信時間も短縮される傾向が見て取れ，生じる通信衝突が少なくなるような位置の代替ノードを利用することで通信時間の削減が期待できる．3.2 節にて述べたように，京では送受信ノードの物理 6 次元座標から通信経路を求めることが可能であるため，代替ノードの選択時に予め通信経路を求め，通信経路の共有が少ないノードを選択することで，通信性能の低下を抑えることが可能である．

また，京では低レベル通信 API を用いた通信により，拡張次元オーダールーティングの最初の ABC 軸の移動を制御できる．この機能を用いることで経路の共有を回避し，通信性能を向上させることが期待できる．今後実装して実験を行い，経路の迂回による通信性能について評価を進める予定である．

4. 関連研究

本研究と同じく，予備ノードを確保した耐故障手法として，プロセスのマイグレーションを用いた手法が存在す

ⓒ 2014 Information Processing Society of Japan

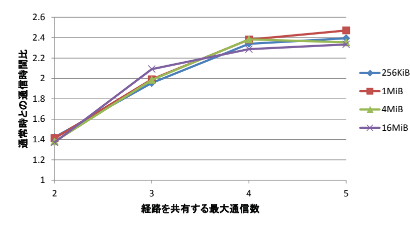


図 11 通信経路共有数ごとの通信時間比較 (6 × 12 ジョブ)

る [5], [6].

このうち，Wang らによる研究 [6] では，故障時にプロセスを移動するのではなく，ノードの状態を監視して故障を予測し故障する前にプロセスを別のノードへと移動して実行を継続する手法が提案されている．この手法により，実行中のジョブが故障に遭遇する確率を低減させることが可能であり，チェックポインティングやリスタートの頻度を減少させることができる．論文では 70%の精度で故障の事前検知ができた場合，チェックポインティングの回数を半分に減らすことができるとされており，大規模環境への適用が期待されている．しかしこの論文において，移動先のノードの決定手法は，利用可能な予備ノードの中で故障予測がされていないものを選択するとされ，ネットワーク上の配置は考慮されていない．そのため，言及されているのはマイグレーションに要するオーバーヘッドのみであり，我々の研究とは異なり移動後の通信性能の変化に関する議論はされていない．

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，2 次元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，代替ノード利用手法を提案し，その通信性能の評価を行った．代替ノード利用手法では，故障したノード上に配置されていたプロセスの実行を，代替ノード上で再開する．その際，代替ノード上のプロセスはネットワークトポロジ的に離れた位置に配置されるため，故障前は存在しなかった経路を共有する通信が生じる．経路上に複数の通信が存在することで，通信の衝突が発生して通信時間が増加する可能性がある．京を用いた実験の結果では，故障ノードが 1 台の場合，代替ノードを用いることで通信時間が最大 2.5 倍になることを確認した．通信時間が増大する原因の一部はまだ調査中であるが，通信の衝突数が大きく影響していることを確認し，衝突の発生しない経路を利用する代替ノードを選択することで，性能低下を抑えられる可能性を示した．

今後の課題として，複数ノード故障時の通信性能の調査が挙げられる．複数ノードが故障すると，さらに通信の衝突数が増加することが考えられるため，故障台数による性能の変化の検証を行い，高性能な代替ノード割り付け手法の検討を進める．また，今回は一対一の通信のみが存在する 2 次元ステンシル通信パターンを対象としたが，今後は集団型通信における通信性能についても調査を進める．そして，一対一通信と集団型通信が混在する実アプリケーションに対して，有効な故障対策手法による耐故障性の実現を目指す．

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] 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).

[5] Chakravorty, S., Mendes, C. L. and Kale, L. V.: Proactive Fault Tolerance in MPI Applications Via Task Migration., *HiPC* (Robert, Y., Parashar, M., Badrinath, R. and Prasanna, V. K., eds.), Lecture Notes in Computer Science, Vol. 4297, Springer, pp. 485–496 (2006).

[6] Wang, C., Mueller, F., Engelmann, C. and Scott, S. L.: Proactive Process-level Live Migration in HPC Environments, *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, SC '08, Piscataway, NJ, USA, IEEE Press, pp. 43:1–43:12 (online), available from ⟨http://dl.acm.org/citation.cfm?id=1413370.1413414⟩ (2008).

ⓒ 2014 Information Processing Society of Japan