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

吉永 一美[1,a)]　堀 敦史[1,b)]　石川 裕[1,c)]

概要：エクサの時代が近づくにつれ、耐故障性の研究の重要性が高まっている。本論文では、メッシュ／トーラスネットワークトポロジーにおいて、ステンシル計算を対象に、ノード故障から復帰するためにスペアノードをどのように割り当て、故障ノードをどのように代替すべきか、という点について研究した結果を報告する。ステンシル計算においては、故障ノードに割り当てられたサブタスクを他の正常なノードに配分する方式は不適切であることを示し、スペアノードによる代替方式を提案する。この時、スペアノードに代替させることで通信メッセージの衝突が発生し、通信遅延時間が増大する可能性があることを示す。本稿では 3 種のスペアノード置換手法を提案し、主にこのメッセージ衝突数という尺度で評価し、それぞれの手法の特徴を明らかにする。結果として、これら 3 つの手法を組み合わせるのが最適と考える。

A Study of Communication Performance When Using Spare Nodes to Replace Failed Nodes

Kazumi Yoshinaga[1,a)]　Atsuhi Hori[1,b)]　Yutaka Ishikawa[1,c)]

Abstract: Towards Exa-scale computing era, the need of fault tolerance research is getting higher and higher. In this paper, how spare nodes should be allocated for stencil programs to replace faulty nodes is investigated. Firstly, it is shown that the way for stencil programs to distribute the workload allocated to the faulty node is not suitable. Then, three methods of how faulty nodes should be replaced by spare nodes are proposed. By using spare nodes, there is a possibility of having communication message collisions, and thus, communication latency can be increased. The three methods are investigated and compared with this maximum number of message collisions to reveal thier characteristics. As a result, the combination of these three methods is considered to be the best.

1. はじめに

ポストペタスケールからエクサスケールへと並列計算機が高性能になるにつれ、大規模となり、故障率の増加が懸念されている [7]。耐故障性の技術として広く研究や応用が進んでいるシステムレベルのチェックポイントは [4]、アプリケーションの変更の必要がないが、状態を退避するために I/O に過大な負担をかけ、エクサスケールでは故障率に対してチェックポイントに要する時間が現実的でなくなることが予測されている [2], [7]。この予測を受けて、チェックポイント時に発生する各種オーバヘッドを低減しようとする研究は数多い（例えば [9]）。一方、アルゴリズムレベルで耐故障性を実現する手法も提案されている [5], [8]。この手法では、チェックポイントに伴う I/O の負荷を低減させ、特にエクサスケールにおいて有効な耐故障を実現するための技術として注目されている。残念ながら、現在広く用いられている MPI 通信ライブラリ [11] の現行規格では、故障発生時の挙動が定義されていないため、アルゴリ

[1] 独立行政法人理化学研究所計算科学研究機構
RIKEN AICS
[a)] kazumi.yoshinaga@riken.jp
[b)] ahori@riken.jp
[c)] yutaka.ishikawa@riken.jp

ズムレベルの耐故障性を実現することができない。

このような現状を踏まえ、故障時の挙動を定義し、通信の途中においてノード故障が発生した際、MPI で記述されたプログラムが故障を検知でき、かつ、その対処が可能となるような MPI ライブラリ User-Level Failure Mitigation（ULFM）が提案されている [1]。ULFM は次期の MPI 標準規格として採用される見通しである。しかしながら、ULFM は、ユーザプログラムが耐故障を実現できるために最低限の機能を提供するにとどまる。このため、耐故障機能を有するユーザプログラムの開発がより容易となるべく、ULFM の上位に位置するライブラリやフレームワークの研究開発も進められている [12], [18]。

本稿では、例えば ULFM を用いたとして、ノード故障が発生してもジョブの継続が可能だったときに、どのように復帰すべきかという点に着目する。科学技術計算において幅広く使われているステンシル計算を対象に、スペアノードの必要性を議論し、どのようにスペアノード群を割り当て、どのスペアノードを故障ノードと代替するのか、その時に通信性能がどの程度劣化するか、といった点に焦点を絞る。本研究はエクサスケール時代における耐故障技術の一端を担うことも目的としている。

2. 関連研究

Falanx[18] は、マスターワーカーモデルにおける耐故障のフレームワークであり、ノード故障時にそのノード上のタスクを別なタスクに割り振ることで耐故障を実現している。Falanx は、計算モデルをマスターワーカーモデルに絞ることでスペアノードを不要としている。

Local Failure Local Recovery（LFLR）[12] は、ノード故障時にそのノードで実行していたタスクを、予め確保しておいたスペアノード上で継続させるというフレームワークである。LFLR は、Falanx より幅広い計算モデルを対象としている。しかしながら、復帰のためスペアノードの存在を仮定してはいるが、どのようにスペアノードを割り当てるかという点に関しての議論はない。

Global View Resilience（GVR）[10] は、通信機能を含む Global View ベースの PGAS ライブラリである。大きな特長としては、配列をバージョニング管理できることで、このため、ユーザが自分でバージョニング（あるいはチェックポイント）の詳細なコーデイングが不要になる。これは、換言すれば、故障からの復帰の仕方についてはライブラリ任せであり、ユーザが細かく制御することができない。このため、全てのクラスのアプリケーションに対し効率的な故障復帰が可能かどうかはライブラリの実装次第である。

Domke らはネットワークのコンポーネント（リンクあるいはスイッチ）が故障した際にそのコンポーネントを使わないようなルーティングをした時に、ネットワークトポロジーやルーティングアルゴリズムの違いにより、どの程度の通信性能の低下が生じるかについて報告している [6]。

3. 貢献

これまでノード故障からの復帰手法のひとつとしてスペアノードを用いる方式が知られていたが、スペアノードを用いることで通信性能に影響がおよぶ可能性があること、その影響を最小限にし、かつ複数のノード故障に耐えるにはどのようにスペアノードを用いれば良いかという点に関する議論はされていなかった。

本稿において、ノード故障の発生から復帰までは、例えば ULFM のような故障レジリエンスのフレームワークの存在を仮定している。本稿では、チェックポイントあるいはアルゴリズム的な工夫による故障復帰が可能であることを前提とし、故障ノードをスペアノードと代替する際、どのような方式が考えられるか、方式により通信性能（メッセージ衝突数）にどの程度の影響を与えるかについて、シミュレーションをベースに調査をおこなった。

本稿では、ステンシル計算におけるノード故障において、故障ノードをどのようにスペアノードと交換すべきかという点に着目し、通信性能が低下する可能性という側面から論じる [14], [15], [16]。我々の知る限りにおいて、このような観点による研究は初めてであり、エクサスケールのコンピュータの実現に寄与すると考える。

本稿の対象をステンシル計算とした理由は、ステンシル計算が大規模シミュレーションにおいて頻繁に用いられているからである。また、ネットワークについては、「京」の Tofu ネットワークのような 2 次元あるいは 3 次元トーラス／メッシュトポロジーを基本とするようなネットワークを想定している [17]。これはエクサ規模のシステムでは、トーラス／メッシュのようなトポロジーが向いていると考えたからである。ルーティングに関しては標準的な XY(Z) ルーティング [13] を想定する。以降、特に断りがない限り、説明を簡単にするため 2 次元メッシュネットワークにおける、5 点ステンシル計算を用いて説明する。また、ノード故障がネットワークに影響を与えないと仮定している。これはつまり、ノード故障が発生しても、例えばネットワークに故障ノードを迂回する機能がある場合、故障発生前と同様に通信が可能であると仮定する。

4. 復帰手法の検討

4.1 しわ寄せ法

ひとつのノード故障が発生した場合（図 1 にノード 21 が故障した例を示す）、そのノードの計算をなんらかの方法により、他の正常なノードに代替させることでジョブが継続可能となる。本稿ではこの方法を「しわ寄せ法」と呼ぶ [15]。マスターワーカーモデルのように、通信パターンが比較的単純で、かつ、並列化アルゴリズムそのものに負

荷をバランスさせる機能を持っている場合は、単純に、故障ノードに割り当てられていたサブタスクを他の正常なノードで実行すれば良い。

ステンシル計算は単純な通信パターンを持ち、問題サイズに応じて適切なノード数を選択することで負荷をバランスさせている。ノード故障が発生した場合、そのノードで実行していたサブタスクを、負荷バランスを保ったまま他の正常ノードに分散させることが難しい。たとえ負荷をバランスさせることができたとしても、元々の単純な通信パターン（5 点ステンシル計算では、東西南北に隣接するノードとの通信）を保つことができない。このため、もともと東西南北方向に位置するノードとだけ通信するように書かれていたプログラムを大きく変更しなければならなくなる。この結果、アプリケーションの開発コストが増大し、かつ、正常時とノード故障が発生した場合でのプログラムコードが大きく異なり、バグの混入を招き易くする。

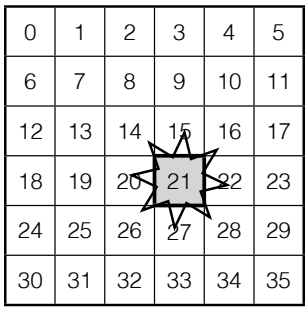

図 1　ノード故障

4.2　スペアノードを用いた復帰方法

一方、スペアノードを予め用意しておき、故障ノード上のサブタスクをスペアノード上に移動させることでジョブの継続が可能になる。アプリケーションが MPI を用いて書かれていた場合、正常時の実行は、スペアノードを除いたノード群を包含するコミュニケータを用い、故障発生時には、故障ノードを除き、代替するスペアノードを含む新しいコミュニケータを作って実行することができる。こうすることで、ジョブの実行に用いる物理的なノード群は故障前と故障後で異なるが、アプリケーション的には違うコミュニケータを使うだけで、計算コードの部分は変更が不要になる。アプリケーションプログラムでは、故障発生時のサブタスクのスペアノードへの移送と、コミュニケータの変更を、ハンドラの中で記述するだけで済む。また、負荷バランスも故障前と後で同様に保たれる [15]。

図 2 に、2 次元メッシュ／トーラストポロジーネットワークにおけるスペアノード群の割当方法を示す。図中“2D-1”とあるのは X（Y）次元方向の大きさを 1 増やして、増えた分をスペアノード群とする方法、”2D-2” とあるのは、X 方向および Y 方向のそれぞれの大きさを 1 増やし、増えた分をスペアノード群とする方法である。3 次元ネットワークの場合も X, Y, Z 方向の大きさを増やすことでスペアノード群を割り当てることができる。メッシュトポロジーでステンシル計算が非周期境界の場合、この方式で通信性能のロスは発生しない。トーラスあるいは周期境界の場合、スペアノード群をまたぐ通信において通信メッセージのホップ数が 1 増える。

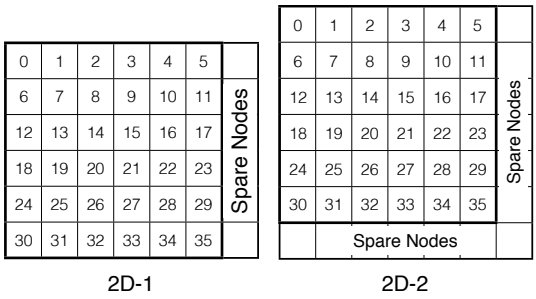

図 2　スペアノード群の割当

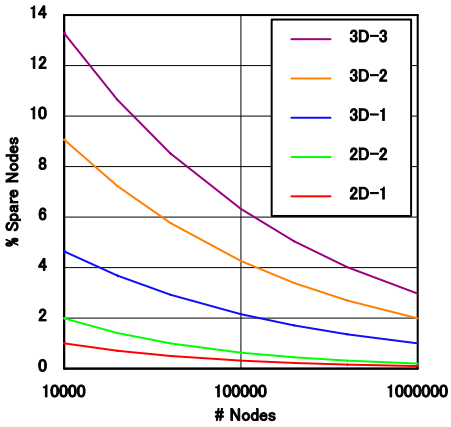

図 3　ネットワークトポロジーと代替ノード数の関係

図 3 は、ジョブ実行のために確保したノード数に対し、スペアノード群がどの程度の割合を占めるか、計算式によって得られた結果をグラフにしたものである。図中、”2D-1” および”2D-2” は先に説明した通りで、2D 割当の時に、スペアノード群を 1 辺とするか 2 辺とするかの違いである。”3D-1” とあるのは 3 次元割当の時に、3 次元の 1 面をスペアノード群、”3D-2” は X、Y、Z 軸の片側 2 面をスペアノード群、”3D-3” は X、Y、Z 軸の片側 3 面をスペアノード群とすることを示す。ちなみに、2 次元の場合にスペアノードは最大 4 辺、3 次元の場合に最大 6 面まで持つことができる。スペアノード群の次元数は、必ずネットワークトポロジーの次元数よりも少ないため、特にノード数が大きくなると、スペアノードの割合が低下する。またトポロジーの次元数が小さい程、スペアノードの割合が減る。図 3 から、ノード数が 10 万ノード規模になるとスペアノードの割合が数パーセント程度となることが分かる。

図 4 は、スペアノードを用いた時の 5 点ステンシル計算

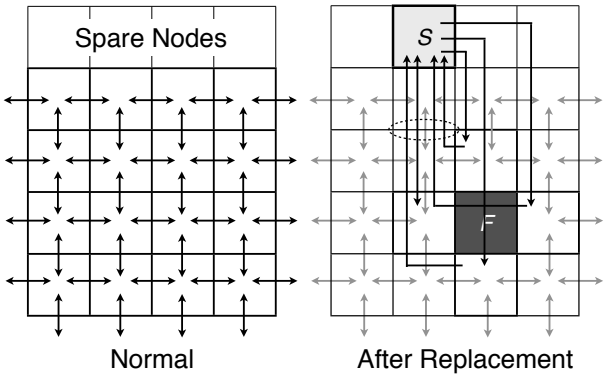

図 4 通信メッセージの衝突の例

における通信メッセージの衝突が発生する様子を示している。左側の図は故障が発生していない通常の通信パターンを示している。右側はノード故障（“*F*” と示されたノード）が発生し、スペアノード（“*S*” と示されたノード）に、故障ノードに割り当てられたサブタスクを移送した時の通信パターンを示している。移送後の通信では、幾つかのメッセージが同じ経路を通るため、この部分でメッセージの衝突が発生し、通信遅延の増大が発生する可能性がある。この図において、楕円で示した部分は、5 つのメッセージが同一経路を通るため、遅延は最大で 5 倍になる可能性があることが分かる。メッセージ衝突数が実際の通信遅延時間に具体的にどのような影響を与えるかについては [15] を参照されたい。

このようにスペアノードに故障ノードを代替させた時にメッセージの遅延が発生する可能性がある。では、遅延を少なくするようなスペアノードの選択にはどのような方法があるのだろうか？図 5 に本稿で提案する 3 つの方式を示す [16]。この図では、21 番のノードが故障したときのスペアノードの使い方を示している。太字のノード番号は、スペアノードの利用に際し、サブタスクが移動したノードを示している。以下、そのそれぞれについて簡単に説明する。詳細に関しては、[16] を参照されたい。

4.2.1 0D Sliding 法

単純に故障ノードをスペアノード群の中のひとつのスペアノードと入れ替える方式で、図 4 に示したものと同じである。この方式では最大スペアノードの数だけのノード故障に対応可能である。しかしながら、複数のノード故障におけるメッセージ衝突数を抑えるためには、スペアノードの選択に配慮する必要がある [16]。

4.2.2 1D Sliding 法

故障ノードを含む行（または列）において、その行（または列）にあるスペアノードを選び、故障ノードからそのスペアノードに至る全てのノードをスペアノードの方向にひとつ移動する。この方式では 1 ノード故障時のメッセージ衝突回数が 3 となり、0D Sliding 方式に比べ通信時間の遅延が少なくなる。この方式の欠点は、スペアノードが

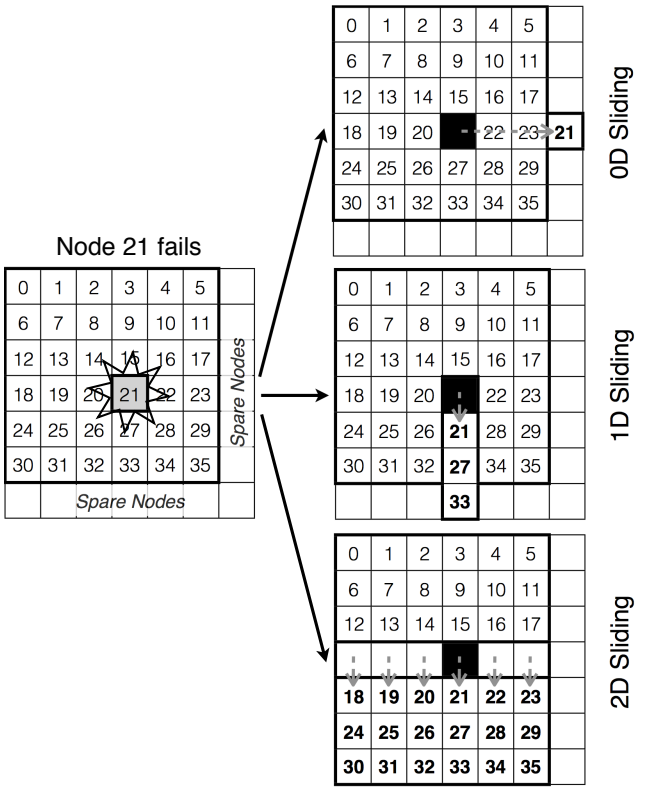

図 5 3 つの復帰手法

2D-1（1 辺）の場合、同じ行（あるいは列）にさらなる故障が発生すると対応できない。この欠点は 2D-2 （2 辺）スペアノードを割り当てることで、少なくとも 3 つのノード故障までに耐えられるよう改善することが可能である。

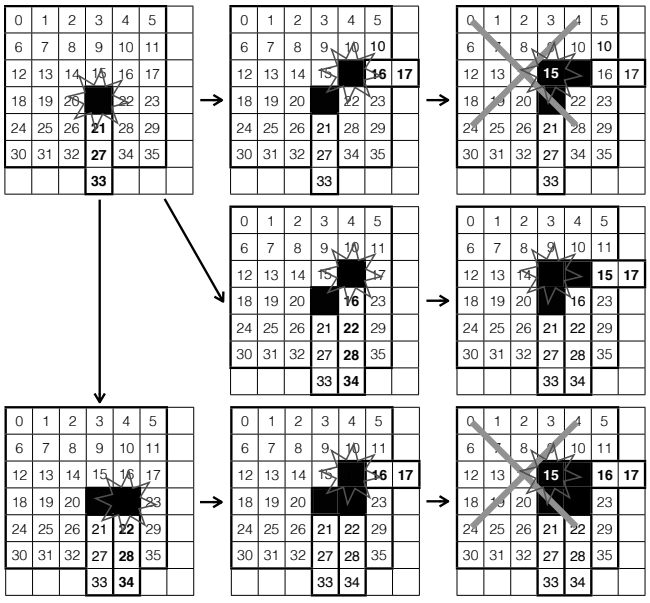

図 6 1D Sliding における代替の例

図 6 は、1D Sliding 方式により複数のノード故障の代替ノードを割り当てた例である。左上に示すようにノード 21 が故障し、下方向にスライドした状態から始まる。最初に上段の故障連鎖について説明する。引き続きノード 16 が

故障し、右にスライドしたとすると、ノード 15 の故障に対応できなくなり、結果として 2 つまでのノード故障にしか対応できていない。2D-2 のスペアノード割当における 1D Sliding では、X 軸方向にスライドするか Y 軸方向にスライドするか、選択が可能である。この状況は、中段に示すように、出来る限り最初に割り当てた方向と同じ方向に割り当てることで回避できる。しかしながらこの方法を用いても、図の下段に示すように、最悪のケースでは、3 つまでのノード故障にしか対応できない。

4.2.3 2D Sliding 法

故障ノードを含む行（または列）にある全てのノードを使わなくする方法である。この方式では、メッセージの衝突は発生しないが、2D-1 のスペアノード割当で最大 1、2D-2 で最大 2 までの故障にしか耐えることはできない。

4.2.4 各手法の比較

図 7 に、故障ノード数と最大メッセージ衝突数（1 は衝突が発生しないことを示す）の関係を各手法とスペアノード割当について示す。この図は、指定した故障ノード数で起きうる全てのノード故障ケースについて、故障後のメッセージ衝突数を計算するプログラムを作成し、それにより得られた故障ノード数ごとの最悪値をグラフにしたものである。

また表 1 に、手法とスペアノード割当における最大対応可能なノード故障数の関係を示す。これらのグラフと表から、大まかに言えば、スライドする次元が大きい程（0D Sliding より 1D Sliding、1D Sliding より 2D Sliding）、また、スペアノードの割当が多い程（2D-1 より 2D-2）最大衝突数が減る傾向にあることが分かる。

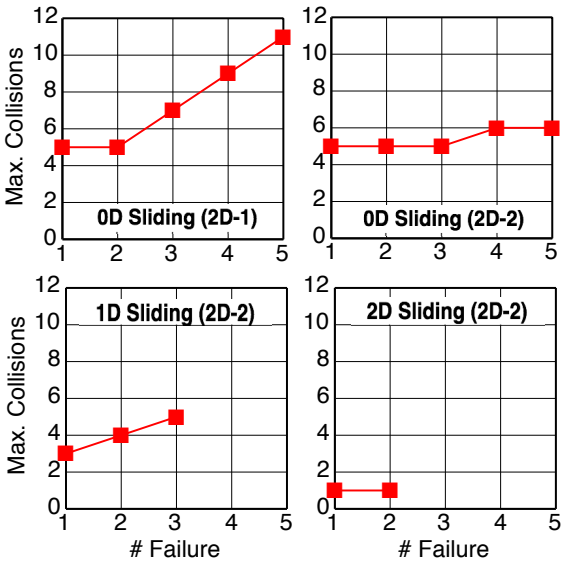

図 7 故障ノード数と最大メッセージ衝突数の関係

4.3 各方式の併用

以上を踏まえ、複数のノード故障において各方式を併用する方式を考える。基本的には、最大メッセージ衝突数が小さい方式を先に適用し、新たな故障が発生した場合にその方式での対応が不可能であった場合に、次の方式を適用する、という併用方式である。

表 1 各方式の比較

方式	スペア割当方式	対応可能な故障回数の上限
0D Sliding	2D-1	スペアノード数
0D Sliding	2D-2	スペアノード数
1D Sliding	2D-1	1（最悪ケース）
1D Sliding	2D-2	3（最悪ケース）
2D Sliding	2D-1	1
2D Sliding	2D-2	2

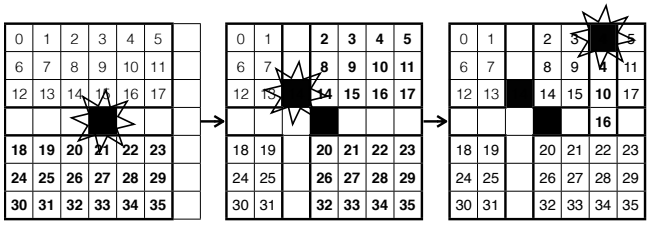

図 8 2D Sliding と 1D Sliding の併用

図 8 にこの併用方式の適用例を示す。この図では、スペアノードを 2D-2 で割り当ててあるとする。左の図は、最初にノード 21 が故障し、2D Sliding を適用した状態を示す。次にノード 14 が故障し、同様に 2D Sliding を適用したのが真ん中の図である。右の図は、さらにノード 4 が故障した時に、以上に加え 1D Sliding 方式を適用した図である。

このように、2D Sliding 方式は、その名が示す通り計算ノード群を 2 次元で移動する。この時、多くのスペアノードが使用されてしまうが、それまで通常の計算ノードであった部分が新たに空き（計算に関与しないノード）となり、これらの空きを新たなスペアノードとすることができる。

4.4 ノード数とメッセージ衝突数の関係

これまでは、ノード故障発生時にスペアノードに代替した時のステンシル計算において、全てのノード故障ケースの中で最大となるメッセージ衝突数に着目してきた。ここでは、各ノード故障ケースでのメッセージ衝突数について、その頻度に着目する。図 9 に、0D Sliding と 1DSliding 方式における単一ノード故障時のメッセージ衝突数の頻度を、プログラムにより求めた値を示す。横軸は 2 次元のノード割当を示しており（例えば “40x40” は 1,600 ノードからなる正方のノード空間、を示す）、縦軸は各ノード割当について求めたメッセージ衝突数の頻度を示す（ノード数の違いを比較できるよう正規化した値となっている）。

この図から、最大メッセージ衝突数がノード数に依存し

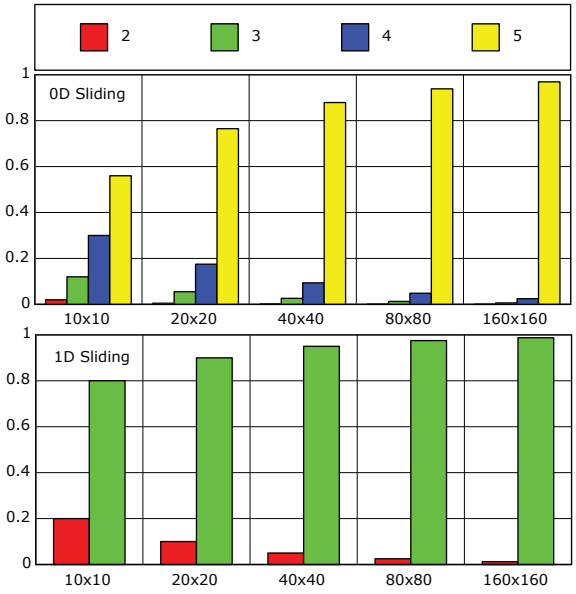

図 9 メッセージ衝突数のヒストグラム

ないこと、さらに、0D Sliding と 1D Sliding の双方において、ノード数が増えるにつれ最大メッセージ衝突数となるノード故障ケースが大半を占めることが分かる。特に 160x160 ノードの場合、ほとんどのノード故障ケースにおいて最大メッセージ衝突数になっている。これは、スペアノード群に隣接するノードが故障した場合や、境界領域を担当する端のノードの場合に衝突数が減るからである（この実験は、非周期境界、メッシュネットワークを条件としている。周期境界、トータルトポロジーの場合については、次章において議論する）。これらのノードは、基本的に 2 次元のノード空間の外周に相当する。N^2 のノード数の場合、外周のノード数のオーダーは $O(N)$ であり、それ以外のノード数のオーダーは $O(N^2)$ となる。このため、N が大きくなるにつれ、最大衝突数となる比率が増えることになる。

5. 議論

5.1 ホップ数への影響

ステンシル計算における通信の大半は隣接ノードとの通信であり、メッシュ／トーラスネットワークにおいてほとんど 1 ホップで通信が可能である。しかしながら故障ノードをスペアノードで代替することでホップ数が増大し、これに起因する通信遅延の増大が懸念される。京コンピュータでは、直接網のルーターを経由するのにおよそ 100ns 程度かかる。これは 100 ホップで 10us となり、無視できない値になる。

0D Sliding 方式では、故障ノードと代替するスペアノードの距離（ホップ数）が大きくなる可能性が高い。一方、1D Sliding や 2D Sliding により増加するホップ数は高々 1 であり、ホップ数増大による通信遅延の増大は無視できる。

5.2 同時送受信数との関係

これまで、5 点ステンシル計算を対象とし、かつ、ステンシルの通信は 4 方向（東西南北）同時におこなうことができると仮定していた。例えば京コンピュータの場合、Tofu のインターフェイスは最大 4 方向にメッセージを同時に送受信することが可能である [19]。同様に BG/Q でも同様に複数のメッセージを送受信することが可能である [3]。このように、最新のスーパーコンピュータでは複数のメッセージを同時に送受信可能になっているものが多い。我々は、エクサクラスのスーパーコンピュータも同様に複数のメッセージを送受信できると仮定し、本稿の大半はこの仮定に基づいている。

1 方向にしかメッセージを送受信できない場合には、大まかに言えば、ネットワークに同時に存在するメッセージ数が減るため、メッセージが衝突する可能性は低くなると考えられる。また、ステンシルの 4 方向の通信を完了するために、1 方向の通信の 4 倍の時間が必要となる。そのため、例えば 4 方向の同じメッセージ長の通信をして、そのうちのひとつの通信時間が倍になったとしても、全体の通信時間は 5/4 程度にしかならない。

一方で、原理的に、例えば 4 方向同時通信可能であるとしても、実際の通信時間が 1 方向のみの通信と同じ時間で終了するとは限らない。例えば、京コンピュータにおける 4 方向同時送信時間は、1 方向のみの場合に比べ 1.7 倍程度となっている。この結果、通信衝突数が 5 となっていても実際の通信遅延時間は 3 倍程度にしかならない [15]。

5.3 メッセージ長の影響

本稿で提案したスペアノードによる代替方式の比較の尺度として、メッセージ衝突数を用いてきた。これはネットワークの同一経路を経由するメッセージの数であり、実際に通信プログラムを走らせて計測した値ではない。実際には、OS jitter や、計算時間のバラツキなどの影響により、各ノードが同時にメッセージを送出するとは考え難い。通信メッセージがネットワーク中に存在する時間よりもメッセージ送出時間のバラツキが十分大きいようなケースでは、メッセージの衝突が起こり難く、結果として通信性能に悪影響は及ぼさない可能性がある。

5.4 9 点ステンシルの場合

これまでの議論は全て 5 点ステンシル計算を対象としていた。5 点ステンシル以外のステンシル計算、例えば 9 点ステンシル計算において、これまでの議論はどのように変化するであろうか？ 9 点ステンシル計算では、5 点ステンシル計算の 4 方向（東西南北）の通信に加え、斜め方向の 4 方向（北東、北西、南東、南西）が新たに加わる。しかしながら、これらの斜め方向の通信量は多くの場合、東西南北方向に比べ非常に小さい。このため、通信時間の大半

は東西南北の 4 方向が占め、斜め方向の通信衝突の影響は無視できると考える。

5.5 周期境界の場合

これまで主に非周期境界のステンシル計算を例にとり説明してきた。図 9 における最大衝突数が小さいケースは、主にスペアノードに隣接したノードが故障した場合と、境界の計算を担うノードが故障した場合であることは先に述べた通りである。周期境界の場合、境界部分を担当するノードが無くなるため、その分、メッセージ衝突数の平均が大きくなると予想される。しかしながら、図 9 で示したように、この効果は規模が大きくなると無視できる程度しかない。

5.6 トーラストポロジーの場合

トーラストポロジーのネットワークにおいては、代替するスペアノードと故障ノードとの距離（ホップ数）がその次元の半分であるような場合、故障ノードの両隣のノードとスペアノードの通信が同じ軸の逆方向に分散されるため、この場合の最大メッセージ衝突数が減ることになる。また、スペアノードに隣接するノード数も増える。これらの理由により、上記にある周期境界の場合とは逆に、メッセージ衝突数の平均が下がると考えられる。また、この効果は規模が大きくなると無視できる程度しかない。

5.7 3 次元ネットワークの場合

これまでの議論では簡単化のため、2 次元ネットワークで議論していた。3 次元ネットワークの場合、3D Sliding 方式が新たに可能になる以外、基本的な特性は同じと考えられる。これは XYZ ルーティング方式の特徴として、最初の X 軸方向のルーティングにより YZ 面に集約され、結果的に 2 次元（YZ 面）の場合と同様に考えることができるからである。

5.8 再配置

複数のノード故障が発生した時に 1D Sliding 方式を繰り返すことで、ノード番号が入り乱れ、結果としてメッセージ衝突回数が増える可能性がある。これを回避するために、例えば 1D Sliding をある回数適用した場合、ノード番号を並べなおす（これはつまり計算に必要なデータを移動することを意味する）ことでメッセージ衝突回数を減らすことが可能と考えられる。残念ながら、現時点でどのようなアルゴリズムでノード番号を振り直すのが最適かは分かっていない。

5.9 小規模ジョブの場合

小規模ジョブが確保するスペアノードによる、システム利用効率の低下は無視できない問題である。一般的なスーパーコンピュータシステムは多数のユーザによって共有され、複数のジョブがシステム内で同時に実行される。図 3 において、ジョブが利用するノード数の増加と共にスペアノードの割合が低下することを示した。100 万ノードのシステムで全ノードを利用するジョブを実行した場合、3D-3 のスペアノード割当では全体の約 3 パーセントがスペアノードとなる。一方、1 万ノードを利用するジョブが 100 個同時に実行された場合、10 パーセント以上のノードがスペアノードとなる。

ノードの故障率を一定とすれば、ジョブ実行中に故障ノードが発生する確率は利用ノード数に比例する。そのため、故障率に対して十分なスペアノードを確保するという観点では、スペアノードも利用ノード数に比例して割り当てればよい。利用ノード数が多い場合は 3D-3 を、少ない場合は 3D-1 を利用するなど、スペアノード割当を利用ノード数によって切り替えることで、近似的に対応可能であると考えている。

6. まとめと今後について

本稿では、エクサの時代を念頭に、ステンシル計算におけるユーザレベルでの耐故障性を実現するための手法としてスペアノードの利用方法に焦点をあてた。故障ノードをスペアノードで代替する方式を選んだ理由は、特にステンシル計算において、故障ノードに割り当てられていたサブタスクを、正常な他のノードに配分する方式では、プログラムに非常に大きな変更が必要となるからである。

スペアノード群の割当方法と、スペアノードを具体的にどのように故障ノードと代替するかについて、0D Sliding 方式、1D Sliding 方式、2D Sliding 方式、および、それらの組み合わせ方式を提案した。スペアノードを用いることで、それまで発生していなかった通信メッセージの衝突の可能性を指摘し、提案手法のそれぞれについて、通信メッセージの衝突数について調査し、比較した。一方、提案手法により対応可能な故障ノード数の上限があることを示した。

提案手法それぞれの最大メッセージ衝突回数、対応可能な故障ノード数を比較検討した結果、最初に 2D Sliding を試み、適用できなくなったら 1D Sliding を試み、最後に 0D Sliding を適用するという組み合わせ方式が、メッセージのホップ数や衝突回数、適用可能な故障ノード数という観点から最適であると考える。

本稿が対象としたスペアノードの割当と利用方法に関する研究は、本稿が最初であり、研究は始まったばかりである。このため、追求すべき課題は数多く残されている。以下にそのいくつかを列挙する。

- ステンシル通信パターン以外でのスペアノード割当方式の検討と評価
- 実アプリケーションを用いた評価

- メッシュ／トーラス以外のネットワークトポロジーにおけるスペアノード割当方法の検討
- 故障ノードを迂回するルーティング方式によるメッセージ衝突数への影響

我々は現在、ステンシルの実アプリケーションを用いた評価を実施しており、この結果については別な機会に報告する予定である。

謝辞 本研究の一部は、科学技術振興機構 (JST) の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」の援助を受けて実施された。

参考文献

[1] 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).

[2] 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).

[3] Chen, D., Eisley, N. A., Heidelberger, P., Senger, R. M., Sugawara, Y., Kumar, S., Salapura, V., Satterfield, D. L., Steinmacher-Burow, B. and Parker, J. J.: The IBM Blue Gene/Q Interconnection Network and Message Unit, *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, New York, NY, USA, ACM, pp. 26:1–26:10 (online), DOI: 10.1145/2063384.2063419 (2011).

[4] Chen, Y., Plank, J. S. and Li, K.: CLIP: A checkpointing tool for message-passing parallel programs, *Proceedings of the 1997 ACM/IEEE Conference on SuperComputing (SC'97)*, pp. 1–11 (1997).

[5] Davies, T., Karlsson, C., Liu, H., Ding, C. and Chen, Z.: High Performance Linpack Benchmark: A Fault Tolerant Implementation Without Checkpointing, *Proceedings of the International Conference on Supercomputing*, ICS '11, New York, NY, USA, ACM, pp. 162–171 (online), DOI: 10.1145/1995896.1995923 (2011).

[6] Domke, J., Hoefler, T. and Matsuoka, S.: Fail-in-place Network Design: Interaction Between Topology, Routing Algorithm and Failures, *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '14, Piscataway, NJ, USA, IEEE Press, pp. 597–608 (online), DOI: 10.1109/SC.2014.54 (2014).

[7] Dongarra, J., Choudhary, A., Kale, S. et al.: The International Exascale Software Project Roadmap, White paper, Argonne National Laboratory (2010).

[8] Du, P., Bouteiller, A., Bosilca, G., Herault, T. and Dongarra, J.: Algorithm-based Fault Tolerance for Dense Matrix Factorizations, *SIGPLAN Not.*, Vol. 47, No. 8, pp. 225–234 (online), DOI: 10.1145/2370036.2145845 (2012).

[9] Gomez, L. A. B., Maruyama, N., Cappello, F. and Matsuoka, S.: Distributed Diskless Checkpoint for Large Scale Systems, *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, CCGRID '10, Washington, DC, USA, IEEE Computer Society, pp. 63–72 (online), DOI: 10.1109/CCGRID.2010.40 (2010).

[10] Group, G.: Global View Resilience (GVR) Documentation, Release 1.0, Technical Report TR-2014-13, University of Chicago (2014).

[11] Message Passing Interface Forum: MPI: A Message-Passing Interface Standard Version 3.0, Message Passing Interface Forum (online), available from ⟨http://www.mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf⟩ (accessed 2015-01-12).

[12] Teranishi, K. and Heroux, M. A.: Toward Local Failure Local Recovery Resilience Model Using MPI-ULFM, *Proceedings of the 21st European MPI Users' Group Meeting*, EuroMPI/ASIA '14, New York, NY, USA, ACM, pp. 51:51–51:56 (online), DOI: 10.1145/2642769.2642774 (2014).

[13] Zhang, W., Hou, L., Wang, J., Geng, S. and Wu, W.: Comparison Research Between XY and Odd-Even Routing Algorithm of a 2-Dimension 3X3 Mesh Topology Network-on-Chip, *Proceedings of the 2009 WRI Global Congress on Intelligent Systems - Volume 03*, GCIS '09, Washington, DC, USA, IEEE Computer Society, pp. 329–333 (online), DOI: 10.1109/GCIS.2009.110 (2009).

[14] 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（オンライン），入手先 ⟨http://ci.nii.ac.jp/naid/110009808101/⟩ (2014).

[15] 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（オンライン），入手先 ⟨http://ci.nii.ac.jp/naid/110009776019/⟩ (2014).

[16] 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（オンライン），入手先 ⟨http://ci.nii.ac.jp/naid/110009850783/⟩ (2014).

[17] 宮崎博行，草野義博，新庄直樹，庄司文由，横川三津夫，渡邊貞：スーパーコンピュータ「京」の概要 (2012).

[18] 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，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. 8, pp. 1–8（オンライン），入手先 ⟨http://ci.nii.ac.jp/naid/110009808103/⟩ (2014).

[19] 志田直之，住元真司，宇野篤也：MPI Library and Low-Level Communication on the K computer, *Fujitsu*, Vol. 63, No. 3, pp. 299–304 (2012).