Vol.2012-ARC-202 No.17
Vol.2012-HPC-137 No.17
2012/12/13

パーシャルメッセージロギングを改善する耐故障性実現フレームワーク

轟 侑樹[1,a)]　實本 英之[2,b)]　佐藤 三久[3,4,c)]　石川 裕[1,2,d)]

概要：
近年のスーパーコンピュータにおけるノード数は近年ますます増加の一途をたどっている．それに伴いＨＰＣシステムにおける故障間隔は無視できない程に短くなり，長時間かかるアプリケーションの実行などにおいてはそれが深刻な問題になり得る．そしてその問題を解決するための耐故障性実現機構として連携チェックポインティングやメッセージロギング方式が提案されている．しかし，連携チェックポインティングでは故障発生時に全てのプロセスがチェックポイントまでロールバックするためにリソースの無駄を生じる．そして一方のメッセージロギング方式では，全ての通信ログを取って置かなければならないため，大きな通信オーバーヘッドがかかってしまう．その両方のデメリットを緩和するための方法として，プロセスをいくつかのクラスタに分割し，そのクラスタ内では連携チェックポインティング，クラスタ間ではメッセージロギングを採用するパーシャルメッセージロギングが提案されている．その中で本研究では，実行時の情報を用いて先行研究の問題点を解決し改善するためのフレームワークを提案する．まず先行研究では，プロセスのクラスタリングを行う為に必要な実行時の通信情報が与えられることを前提としているが，本フレームワークでは，アプリケーションのメインのループを指定してもらい，そのループの通信を解析することで適切なクラスタリングを取得する．また，先行研究ではグラフパーティショニングツールを用いているが，グラフバーティニングツールは汎用的なグラフ分割を対象にしており，プロセスのクラスタリングには比較的長い時間を要する．そこで本研究では，プロセスがどのようにノードに割り当てられているかの情報を利用することでより単純で高速にプロセスを分割するアルゴリズムを本フレームワーク上に実装する．そして，単純なステンシル計算をシミュレートするグラフ分割で提案手法の評価を行う．

Fault Tolerance Framework to Improve Partial Message Logging

Yuki Todoroki[1,a)]　Hideyuki Jitsumoto[2,b)]　Mitsuhisa Sato[3,4,c)]　Yutaka Ishikawa[1,2,d)]

Abstract:
Modern supercomputers consists of more and more nodes. That makes mean time between failure (MTBF) shorter than that of current situation. If MTBF becomes less, application that takes much time can not be done because of failure. To cope with the problem, many fault tolerance mechanisms are proposed. Coordinated checkpointing and message logging are typical mechanisms to achieve fault tolerance. However, coordinated checkpointing forces all processes to roll back to last checkpoint when occuring failure and message logging imposes large overhead on communication. Partial message logging is proposed to alleviate those disadvantages. The method splits processes into some groups and takes checkpoints within a group and logs data of communication between groups. Due to this method, the processes which have to be restarted can be limited from all processes to processes within a group. In this research, we proposes a framework to improve partial message logging by analyzing application behavior at runtime. Prior research assumes whole communication pattern of application, so application have to be executed in advance. Our framework analyzes main loop of HPC application which has specified by application programmer. In addition, prior research uses graph partitioning tool to group processes, but graph partitioning tool is designed for generic graph partitioning problems and take longer time to partition processes. Therefore, we proposes simple and faster grouping algorithm on the framework utilize mapping between processes and nodes. Proposed method is evaluated through partitioning communication graph which simulates simple stencil computation.

1. はじめに

HPC 分野において，耐故障性の重要性は以前から指摘されている．エクサスケールに向けて数百万のオーダーのコア数で構成されるシステム [4] において，パーツ点数の増加によって故障率の増加は避けられない問題である．しかしながら，再開，チェックポイントを含めた耐故障性の実現にかかる時間が故障間隔を超えてしまえば，アプリケーションの実行は進まないため，よりコストの低い耐故障性実現手法というものが強く要求されている．耐故障性の実現における一般的な手法として，各実行プロセスが同期を取って一貫性のあるチェックポイントを取る連携チェックポインティングが提案されている．連携チェックポイントにおけるデメリットとしては以前から I/O が連携する瞬間に集中するためそれがボトルネックとなってしまうという問題が指摘されていた．[3] それを解決するためのアプローチとして，チェックポイントのサイズ自体を削減する手法 [12]，非同期にチェックポインティングを取りつつも一貫性を確保するための手法，[2], [9] そして段階的なチェックポイントを取る手法 [1] など，様々なものが提案されている．そのような研究により，連携チェックポインティングの最も大きな問題点として指摘されていた問題は徐々に解決に向かいつつある．しかしながら，依然として連携チェックポインティング手法においては，故障発生時に全てのプロセスが巻き戻らなければならず，同じ処理を繰り返すことによってリソースを無駄にするという点が問題点として挙げられる．もし非同期にチェックポイントを取り，各プロセスが個別に再開できればそれは最もリソースの無駄が少ない．そのような代表的な手法としてメッセージロギングが提案されている．この手法は，プロセス間の通信のログを全て取っておき，再開時にその情報を使って一貫性のある状態を保持する手法であるが，これには全てのメッセージを，その内容も含めてログを取っておく必要があり，通信の際に大きなオーバーヘッドがかかる．この通信時のオーバーヘッドを緩和しつつ，再開するプロセス数を減らすアプローチとして，パーシャルメッセージロギングが提案されている．[8], [11] これは，プロセスをいくつかのグループに分割しておき，その中では連携チェックポインティング，グループの間の通信はロギングを行うことでロールバックするプロセスをひとつのグループの中に限定する手法である．この手法を用いるためには，グループをどのように形成するかが重要になってくる．なぜなら，グループ間の通信が少なければ少ないほどログを取る必要のあるデータが減り，通信時の性能低下が起きにくくなるからである．基本的にはグループを形成する問題は古くから扱われているグラフ分割問題と同様に考えることができる．グラフ分割問題における点をプロセス，辺を通信，重みを通信量と考えエッジカットを最小にするような分割を求めれば良い．そのようなグラフを考えるには，アプリケーションの通信パターンを取得することが必要である．パーシャルメッセージロギングにおける先行研究 [10] では通信パターンを前提として分割を考えていたが，通信パターンを取得しようと思えば事前にアプリケーションを一度実行しておかなければならず実用上それは不可能である．もしくは，小さな問題ケースを用意しておき，それを実行して類似の通信パターンを取得しておくなどのアプローチも考えられるが，ユーザにアプリケーション以外の負担を強いる上に，実際の通信パターンを良く反映する問題ケースがどういうものかなどは自明ではない．そこで本研究では，メインのループをユーザに指定してもらいその挙動を実行時に解析する．多くの HPC アプリケーションでは，実行時間の大部分を占めるメインのループが存在し，その挙動を解析することでアプリケーションの大体の挙動を把握することができる．そしてそのフレームワーク上に，グリッド形状のグラフに対するグラフ分割アルゴリズムを提案する．これは，全ての点を調べる事なしに境界線のみを決定することにより計算量を削減することを目標としたものである．そのアルゴリズムは非構造なメッシュなどには対応できない問題があるが，物理ノードにどのようにプロセスが割り当てられたかといった情報を利用し，物理ノードベースでグルーピングを考えることで多様な問題に対応させることができる．提案アルゴリズムは，4-way の二次元ステンシル計算をシミュレートしたグラフを分割する際の実行時間で評価し，広く使われているグラフ分割ツールである METIS と比較して非常に高速にクラスタリングを行うことができると共によりノード数に対してスケーラブルであることを示した．

2. 関連研究

最も関連の深い研究として、パーシャルメッセージロギングのクラスタリング手法についての改善を行っているものがある。[10] この研究においては、パーシャルメッセージロギングに必要となるクラスタリングアルゴリズムについての提案を行なっている。このアルゴリズムは、MeTiS などのグラフ分割ツールにおける問題点の一つである、分割数を事前に与える必要があるという問題を解決している。

[1] 東京大学大学院情報理工学系研究科
Graduate School of Information Science and Technology, The University of Tokyo
[2] 東京大学情報基盤センター
Information Technology Center, The University of Tokyo
[3] 筑波大学システム情報工学研究科
Graduate School of Systems and Information Engineering, University of Tsukuba
[4] 筑波大学計算科学研究センター
Center for Computational Sciences, University of Tsukuba
[a)] todoroki@is.s.u-tokyo.ac.jp
[b)] jitumoto@cc.u-tokyo.ac.jp
[c)] msato@cs.tsukuba.ac.jp
[d)] ishikawa@is.s.u-tokyo.ac.jp

そして、分割に対する評価関数を定義している。しかしながら、通信パターンは事前に与えられることを前提としており、実用上は事前に同一のアプリケーションを実行するか、小さいテストパターンを走らせるなどして類似の通信パターンを取得する必要がある。我々の研究ではそのような問題を解決することが可能である。また、その先行研究でも用いられているグラフ分割ツール METIS に関する研究が関連として挙げられる。[6], [7] このアルゴリズムは汎用的なグラフ分割問題においては非常に有効である。まず相互に密接に接続されているノード同士を統合してひとつのノードとして扱うことにより、元のグラフよりノード数の少ないグラフに変形させる。そしてそのグラフを単純な Graph Growing Algorithm などで分割し、精度を高めながら元のグラフに変形し直していく、という段階を踏む。そのそれぞれにおいて、E を辺の数とした時 $O(|E|)$ の計算量が必要とされる。本研究で提案しているアルゴリズムは、MeTiS よりも単純にかつ高速にクラスタリングを行えないか、ということを主眼に置いて設計されている。二次元、三次元グリッドのようなグラフにおいて、全ての点を調べることなしに境界線のみを決定することで、ノード数を N とした時 $O(\sqrt{N})$ の計算量で境界を決定することができる。そして、領域分割の分野でも古くからグラフ分割問題に対するアプローチが行われている。[5] Recursive Coordinate Bisection アルゴリズムや、Recursive Graph Bisection などのアルゴリズムは我々のアプローチと非常に近いものである。しかしこれらのアルゴリズムは設計に関しては単純ではあるが、計算量に関しては必ずしも小さくなく、グラフのノード全てをソートする必要性があることなどから、少なくとも $O(n \log n)$ の計算量が必要になるため、計算量の面で提案アルゴリズムは優位がある。

3. 提案手法

3.1 ユーザによるディレクティブを用いた実行時解析

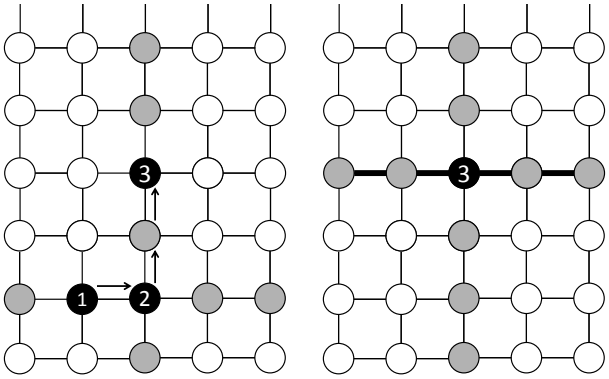
まず，本フレームワークではメインのループの解析を行うことでアプリケーションの挙動を解析する．このようなメインのループは，ユーザから見れば当たり前に知りうる情報ではあるが，フレームワークがそれを検知することはそれほど容易ではない．よって，ユーザにこのループの位置を指定してもらう必要がある．その指定の概要を図 1 に示す．図 1 のように，ディレクティブをメインのループに挿入してもらうことでフレームワークがメインのループの存在を検知する．具体的には，プリプロセッサでこのようなディレクティブを処理し，目的に応じて必要なコードを挿入する事で解析，及びその利用を行う．実行のほとんどがこのループによるものであるため，この内部の通信部分の処理，計算や代入処理を見ることで全体のアプリケーションの処理の傾向をつかむことができる．本研究では，このディレクティブで指定されたループの一周目だけを実行し，その通信部分の解析結果からプロセスのグルーピングを行なっている．

ディレクティブが挿入されたコード例

```
…
#pragma MAIN_LOOP   //メインループの指定
for (i…) {
    …
    MPI_…();        //通信部分
    A[i]… = …;      //計算，代入処理
    …
}
…
```

図 1 ユーザによるメインループ指定方法

3.2 グリッド形状のグラフに対する分割アルゴリズム



(a) 中央の点を探す　(b) 幅，高さを調べ短い方で分割

図 2 分割アルゴリズムの概要

本フレームワークの動的解析で得られた情報を利用して，グラフを分割するアルゴリズムを提案する．このアルゴリズムは，ある程度グラフの形状やメッシュの形状に前提を加えることで，より高速にグルーピングを行うことができる．このアルゴリズムの概要を図 2 に示す．例では二次元の 4-way ステンシル計算における通信パターンを元にしたグラフを分割する．まず適当に選んだ開始ノードから，グラフ全体の中央に近い点を探す．一方向に対して端点に到達するまでノードを辿っていくことでグラフの大まかな幅を測定し，その真ん中のノードに開始ノードから移動する．そして同様に，今度は先ほどの方向に対して直交する方向に同様の処理を行い，高さを測定し，その真ん中に移動する．グラフ全体の形状が，くぼみや穴がない事を仮定できれば，これで真ん中に近い点に移動することが可能である．そしてそこから，同様に高さ，幅を測定し，短い方でグラフを分割する．具体的にノードをある方向に辿る，という事はこの場合に簡単な処理ではない．二次元のグリッド形状のグラフも，各ノードが持っている情報はど

のプロセスと隣接しているか，という情報のみでそれだけで方向を判断しなくてはいけない．そこで本アルゴリズムでは，ノードのホップ数を利用して方向を検知する手法を用いている．その様子を図 3 に示す．図 3 において，P のノードが直前に居たノード，S のノードが現在のノード，D のノードが進みたいノードである．P と S が与えられた時に，S から見て進めるノード三つの中から適切に進むノードを選択したい．そのために，P と S をつなぐ辺を無視した P からのホップ数を考える．その時，D のノードだけはホップ数 2 の範囲に含まれないため，トポロジー情報のない状態でも直進，直交を定義することができる．提案アルゴリズムは，全ての点を調べる必要がなく，境界線の周りのいくつかの点を調べることでグラフ分割を手に入れる事ができる．同様の手法は三次元に対しても，高さの概念を加えることで対応できる．また，いくつかの形状のメッシュにも，縦と横の概念を定義できれば同様のアルゴリズムが適用可能である．しかし，三次元の四面体や非構造なメッシュには対応できないという問題がある．

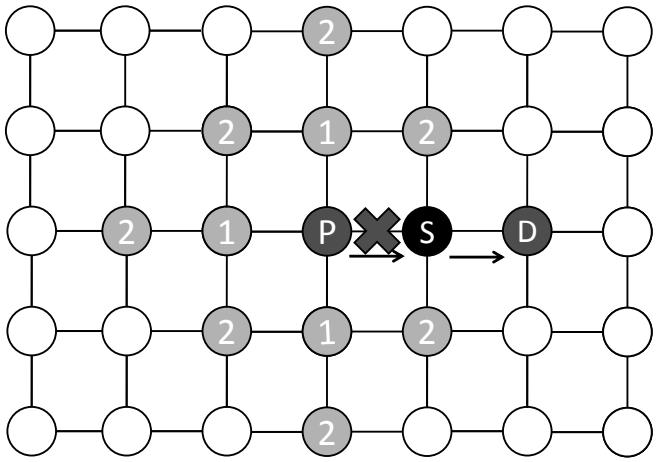


図 3　ホップ数によって方向を決定する手法

3.3　多様な問題への対応

先述のアルゴリズムでは，対応できるメッシュの形状はある程度限られており，特に非構造なメッシュには対応することが困難であったが，それを解決するために，物理的なノードへの割り当て情報を用いるアプローチが考えられる．どのようなメッシュ形状，通信パターンの問題でも実行の前に物理的なノードに計算を分割し，割り当ててから実行される．あらかじめそれらの物理的なノードに対して，二次元，三次元のグリッドなど，先述のアルゴリズムで十分対応できるトポロジーを設定しておく．そして，そのような物理ノードに対して割り当てたプロセスをノード単位で分割し，同一ノードに割り当てられているプロセスは同一のグループとする．こうすることで，多様な問題においても上述の高速なアルゴリズムを適用することが可能になる．もちろんこの手法では，分割の精度は適切に分割され，物理ノードに割り当てられているかに大きく依存するが，本来 HPC アプリケーションが性能を出すためには通信の物理的トポロジーを考えた割り当てが重要になっている．京や Bluegene/Q などの近年のスケーラブルなネットワークを持つスーパーコンピュータでは，二次元や三次元のグリッドなネットワーク構成を効率的にサポートしている．そのため，適切な領域の分割や，通信を考慮したプロセスのノードへの割り当てを前提とすることも可能であると考えられる．

3.4　評価関数を用いた分割数指定の自動決定

提案アルゴリズムは二分割を繰り返す手法であるため，分割を途中で止める事も可能である．何らかの評価関数を与える事で，あらかじめ分割数を与えなくても自動で分割を適切な段階で止めることができる．具評価関数などの体的なとしては先行研究 [10] で提案されているものを用いている．しかし，評価関数も提案アルゴリズムの特徴である境界線の情報を用いることで近似的にではあるが効率的に実装することが可能である．評価関数の算出に必要なものは，各分割のノード数，エッジカットのサイズが主なものとなる．もしグラフ全体の形状が，二次元や三次元のグリッドを仮定できれば，境界線の情報が存在すれば，それによって分割される長方形面積，直方体の面積として分割のノード数が計算できる．そして，境界線が分割する辺の数を考慮することでエッジカットを計算することができる．これらの計算は境界線の情報が利用できれば，各点を逐一確認する必要なく境界線の点のみを考えることで計算できる．

4.　評価

4.1　実験環境，及び条件

表 1　実験環境

CPU	Dual-Core AMD Opteron(tm) Processor 1214 (2.21GHz, 16cores)
Memory	2GB
OS	Ubuntu 11.10(oneiric)
Kernel	Kernel Linux 3.0.0-12-generic x86-64

実験を行った環境を表 1 に示す．現在のところ，実際のアプリケーションを走らせてオンラインで行った実験ではなく，ステンシル計算などをシミュレーションするシンプルなグラフを用意し，グラフ分割だけをオフラインで実行することでアルゴリズムの評価を行った．この実行時間は I/O 時間を除いており，また各プロセスへ分割情報を伝達する時間も含んでいない．

4.2 提案アルゴリズムの実行時間

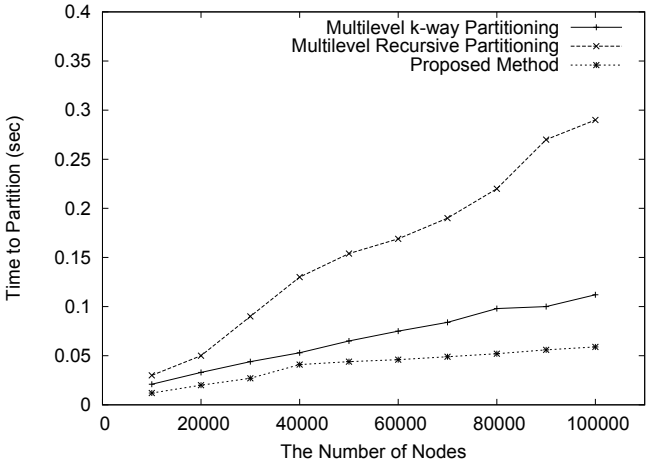


図 4 4-way の二次元ステンシル計算のグラフ分割時間

提案手法におけるグラフ分割時間を，MeTiS で用いられているアルゴリズムと比較した結果を図 4 に示す．グラフの分割数は 100 で固定とし，ノード数による実行時間の変化を測定した．提案手法と Multilevel Recursive Bisection は再帰的に二分割を繰り返すアルゴリズムであり，分割数に対して線形に実行時間が増大する．一方，k-way Multilevel Biesction とは一度の分割でグラフを k 個に分割するため分割数の増大に対してスケーラブルであるが，途中で分割を止めることができない．そのため，もし分割数の見積りが事前にできない場合には分割数を小さい数から順々に試していくことが必要になるので，実用的にはより多くの時間がかかることが予想される．評価関数を用いた分割数の自動決定を提案している先行研究 [10] では，二分割を繰り返してその都度評価関数を呼び出すため，その場合には Multilevel Recursive Bisection と同様の実行時間に加え，評価関数によるオーバーヘッドが存在することになる．図 4 の実行時間の変化を見ると，Multilevel Recursive Bisection や k-way Multilevel Biesction はノード数に対して線形にノード数が増大しているのに対し，提案手法ではノード数の平方根に比例して実行時間が増大していることがわかる．そのため，更に大きなノード数を扱う場合において，提案手法はよりスケーラブルであると言える．

4.3 評価関数による分割数の自動決定を行う場合の性能

また，評価関数によって分割数を自動決定する場合の性能についての実験結果を図 5 に示す．比較対象として，METIS を評価関数と組み合わせる手法を実装し，同様の実験を行った．METIS のライブラリを用いる場合，評価関数と協調させるためにはいくつかの処理が必要になる．分割された二つのグラフを再度 METIS にかけるため，分割結果と元のグラフから新たな二分されたグラフを再構成しなくてはいけない．この処理には，$O(E)$ の計算量がかかってしまい，オーバーヘッドとなってしまう．まず，評価関数に必要な変数を分割結果から数え上げなどを行なって取得しなくてはいけない．一方で，提案アルゴリズムにおいては境界線の情報を利用することにより，$O(1)$ でそれらの処理が行える．実験結果では，評価関数なしとほぼ変わらない結果となっている．これは評価関数を利用するためのコストがグラフを分割するコストよりも小さいため，グラフ分割自体のコストが支配的となっているためである．評価関数無しの場合よりも同じノード数においても実行時間が少ない場合があるのは，再帰的に二分割を繰り返すアルゴリズムでは分割数にも比例して実行時間が増えるのだが，その評価関数によって自動的に決定された分割数が評価関数なしの実験で用いた 100 という定数よりも小さくなっている影響である．

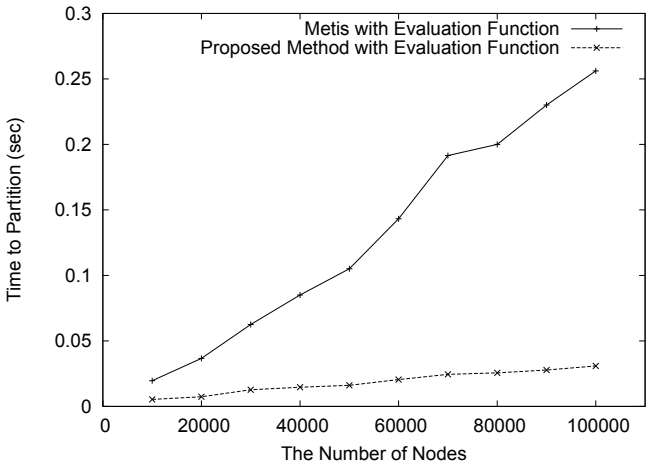


図 5 評価関数で分割数を自動決定する場合のグラフ分割時間

5. 今後の課題

今後，本研究をより進めるためには以下の内容を行うことが重要であると考えている．まず第一に，本論文の範囲ではグラフ分割の質の評価を行なっていない．提案アルゴリズムでは，グラフの形状をある程度想定することにより，点を全てチェックすること無く境界線のみを定めて分割を行う．メッシュ形状が長方形，直方体などと綺麗な形状であるアプリケーションの通信グラフを扱う際には精度も十分に確保できると考えるが，場合によっては精度が犠牲になる場合も存在する．そのため，分割の精度評価を行う必要がある．また，提案手法においては，プロセスへの分割情報の伝送方法が異なる．MeTiS などを用いた場合には分割情報が一つのノードに集約され，それを MPI_Scatter などで全プロセスに渡せば済むのだが，我々の提案手法では境界線をベースに，隣接に各プロセスに分割情報を伝達する必要がある．そのため，その処理も含めて分割手法を評価する必要が生じる．そして，3.3 で述べた物理ノードへの割り当てを考慮した分割を実装，評価する必要がある．

この手法は先述の通り，領域分割や割り当ての精度によって分割精度に差が生じるが，それを十分に最適化した結果を用いて分割速度，精度評価を行う予定である．

6. まとめ

本論文では，ユーザからのディレクティブを用いて実行時解析を行う事でより効率的にパーシャルメッセージロギングを実現するためのフレームワークを提案した．ディレクティブとして，ユーザからメインのループ位置などを指定してもらい，それをプリプロセッサで処理することで挙動の解析，その結果の利用に必要なコードを挿入する．そして当該フレームワークの持つ特徴としての，グラフ分割アルゴリズムの提案を行った．グラフ分割アルゴリズムは，二次元，三次元のグリッドなどの綺麗な形状のグラフに対して全ての点を調べること無く境界線のみを決定することにより既存の手法よりも効率的にグラフを分割することができる．そしてのそのような手法をプロセスの物理ノードへの割り当て情報と物理ノードのトポロジ情報を利用することで多様な問題に対して対応可能であることを示した．提案手法は，4-way の二次元ステンシル計算をシミュレートする二次元グリッドなグラフを分割する性能比較によってその性能，スケーラビリティにおいての優位性を示した．

参考文献

[1] Bautista-Gomez, L., Tsuboi, S., Komatitsch, D., Cappello, F., Maruyama, N. and Matsuoka, S.: FTI: high performance fault tolerance interface for hybrid systems, *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, New York, NY, USA, ACM, pp. 32:1–32:32 (online), DOI: 10.1145/2063384.2063427 (2011).

[2] Bronevetsky, G., Marques, D. J., Pingali, K. K., Rugina, R. and McKee, S. A.: Compiler-enhanced incremental checkpointing for OpenMP applications, *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming*, PPoPP '08, New York, NY, USA, ACM, pp. 275–276 (online), DOI: 10.1145/1345206.1345253 (2008).

[3] Cao, G. and Singhal, M.: On coordinated checkpointing in distributed systems, *Parallel and Distributed Systems, IEEE Transactions on*, Vol. 9, No. 12, pp. 1213 –1225 (online), DOI: 10.1109/71.737697 (1998).

[4] Dongarra, J., Beckman, P., Moore, T., Aerts, P., Aloisio, G., Andre, J.-C., Barkai, D., Berthou, J.-Y., Boku, T., Braunschweig, B., Cappello, F., Chapman, B., Chi, X., Choudhary, A., Dosanjh, S., Dunning, T., Fiore, S., Geist, A., Gropp, B., Harrison, R., Hereld, M., Heroux, M., Hoisie, A., Hotta, K., Jin, Z., Ishikawa, Y., Johnson, F., Kale, S., Kenway, R., Keyes, D., Kramer, B., Labarta, J., Lichnewsky, A., Lippert, T., Lucas, B., Maccabe, B., Matsuoka, S., Messina, P., Michielse, P., Mohr, B., Mueller, M. S., Nagel, W. E., Nakashima, H., Papka, M. E., Reed, D., Sato, M., Seidel, E., Shalf, J., Skinner, D., Snir, M., Sterling, T., Stevens, R., Streitz, F., Sugar, B., Sumimoto, S., Tang, W., Taylor, J., Thakur, R., Trefethen, A., Valero, M., Van Der Steen, A., Vetter, J., Williams, P., Wisniewski, R. and Yelick, K.: The International Exascale Software Project roadmap, *Int. J. High Perform. Comput. Appl.*, Vol. 25, No. 1, pp. 3–60 (online), DOI: 10.1177/1094342010391989 (2011).

[5] I, H. D. S., Simon, H. D. and Simon, H. D.: Partitioning of Unstructured Problems for Parallel Processing (1991).

[6] Karypis, G. and Kumar, V.: A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs, *SIAM J. Sci. Comput.*, Vol. 20, No. 1, pp. 359–392 (online), DOI: 10.1137/S1064827595287997 (1998).

[7] Karypis, G. and Kumar, V.: A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs, *SIAM J. Sci. Comput.*, Vol. 20, No. 1, pp. 359–392 (online), DOI: 10.1137/S1064827595287997 (1998).

[8] Meneses, E., Mendes, C. L. and Kale and, L. V.: Team-Based Message Logging: Preliminary Results, *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*, pp. 697 –702 (online), DOI: 10.1109/CCGRID.2010.110 (2010).

[9] Norman, A. N., Choi, S.-E. and Lin, C.: Compiler-generated staggered checkpointing, *Proceedings of the 7th workshop on Workshop on languages, compilers, and run-time support for scalable systems*, LCR '04, New York, NY, USA, ACM, pp. 1–8 (online), DOI: 10.1145/1066650.1066663 (2004).

[10] Ropars, T., Guermouche, A., Uçar, B., Meneses, E., Kalé, L. V. and Cappello, F.: On the use of cluster-based partial message logging to improve fault tolerance for MPI HPC applications, *Proceedings of the 17th international conference on Parallel processing - Volume Part I*, Euro-Par'11, Berlin, Heidelberg, Springer-Verlag, pp. 567–578 (online), available from ⟨http://dl.acm.org/citation.cfm?id=2033345.2033406⟩ (2011).

[11] Yang, J.-M., Li, K. F., Li, W.-W. and Zhang, D.-F.: Trading off logging overhead and coordinating overhead to achieve efficient rollback recovery, *Concurr. Comput. : Pract. Exper.*, Vol. 21, No. 6, pp. 819–853 (online), DOI: 10.1002/cpe.v21:6 (2009).

[12] Yang, X., Wang, P., Fu, H., Du, Y., Wang, Z. and Jia, J.: Compiler-Assisted Application-Level Checkpointing for MPI Programs, *Distributed Computing Systems, 2008. ICDCS '08. The 28th International Conference on*, pp. 251 –259 (online), DOI: 10.1109/ICDCS.2008.25 (2008).