様　式　Ｃ－１９、Ｆ－１９－１、Ｚ－１９　（共通）

科学研究費助成事業　　研究成果報告書

令和　６ 年　 ６ 月　 ５ 日現在

機関番号：１４３０３
研究種目：基盤研究(C)（一般）
研究期間：2021～2023
課題番号：２１Ｋ１１８０６

研究課題名（和文）スレッドレベル並列性抽出のための動的コードマイグレーション方式の研究

研究課題名（英文）Study on dynamic code migration to extract thread-level parallelism

研究代表者
平田　博章（Hirata, Hiroaki）

京都工芸繊維大学・情報工学・人間科学系・教授

研究者番号：９０２７３５４９

交付決定額（研究期間全体）：（直接経費）　　3,200,000円

研究成果の概要（和文）：並列性を静的に抽出できないプログラムであっても、並列処理による高速化を可能とする方法としてスレッドレベル並列投機実行が有望である。しかし、十分な高速化を達成するためには、投機実行に失敗した場合の影響を可能な限り取り除く必要がある。そこで、本研究では、(i)投機実行の失敗そのものを回避する、(ii)投機実行に失敗しても性能への影響を低減する、の2点において効果を発揮する動的コードマイグレーション方式を新たに開発した。性能評価の結果、本方式によってプログラムの実行時間を劇的に短縮できることを確認するとともに、スレッドレベル並列投機実行においてさらに高速化を実現するための着想も得た。

研究成果の学術的意義や社会的意義
本研究は、スレッドレベル並列投機実行の研究から着想したものであるが、その結果はスレッドレベル並列投機実行の範疇を超えている。コミット処理を要する点で投機実行の特質を残しているものの、投機に失敗しないという点で、従来の非投機的な並列実行の新しい方式とも捉えることができる。このような質的進化は、並列処理研究の新たな局面を切り開くものであり、学術的に大きな意義が認められる。
また、本研究の成果は多くの分野のプログラムに利用可能である。ビッグデータや人工知能で使用するアルゴリズムでも利用可能であり、他の重要な研究分野を含む広い範囲でプログラムの実行時間短縮に貢献できる点で社会的な意義も大きい。

研究成果の概要（英文）：Thread-level parallel speculation is promising for speeding up programs by forcing them to be executed speculatively in parallel, even if it is impossible to analyze it statically and judge whether it is parallelizable. However, to fully extract the inherent parallelism of a program, the effects of misspeculation should be removed as much as possible. So, in this study, we developed a dynamic code migration scheme, which both (i) avoids misspeculation itself and (ii) reduces the penalty even if the misspeculation arises. Newly developed. Through our performance evaluation and analysis of evaluation results, we confirmed that our dynamic code migration scheme can dramatically shorten program execution time, and we could also get a new idea to promote further parallelization and performance improvement.

研究分野：情報工学

キーワード：計算機システム　コンピュータアーキテクチャ　ハイパフォーマンスコンピューティング　スレッドレベル並列処理　投機実行

様　式　Ｃ－１９、Ｆ－１９－１、Ｚ－１９（共通）

１．研究開始当初の背景

(1)　　ハイエンドのサーバからパソコンや携帯電話機、それに小型の組み込み機器に至るまで、複数のプロセッサコアを搭載するマルチプロセッサ型のマイクロプロセッサが使用されるようになっていた。このような状況は現在でも変わらず、そのコア数やマイクロプロセッサ数は当時よりも増加し続けている。しかし、このようなマルチコア化の恩恵を得るためには、スレッドレベルの並列プログラムを作成しなければならない。

(2)　　一方、プログラムの並列化の研究は古くから行われてきた。一般に、スレッド間の依存関係を静的に解析して、スレッド間にデータの依存関係がないことが確かめられれば、それらのスレッドを並列実行することで高速化を図る。依存関係があれば並列実行できないが、実際には、実行前に依存関係の有無を判別できない場合が多い。したがって、依存解析の判定結果には「並列化可能」「並列化不可能」の他に「判定不能」があり、判定不能の場合は並列化せずにあきらめざるを得なかった。しかも、一般的な多くのプログラムでは「判定不能」となるものが非常に多い。

(3)　　そこで、依存関係の有無があらかじめ判別できなくても、依存関係がないものと仮定して投機的に並列実行することにより、プログラム実行の高速化を図るアプローチが注目されていた。それがスレッドレベル並列投機実行である。しかし、プログラム実行の正しさを保証するためには、実行時にスレッド間の依存関係を解析しなければならない。そして、依存関係があることを検出した場合（つまり、投機に失敗した場合）は、そのスレッドの実行結果を破棄して（これを「アボートする」という）、そのスレッドを最初から実行し直す。アボートすることが少なければ、それだけ高速化が見込める。

(4)　　しかし、当時のスレッドレベル並列投機実行における技術的な関心は、実行時の依存解析の実現方式やアボートした場合の再実行の仕組みのみに集中していた。並列化をあきらめていた状況と比較すれば、単に並列投機実行を試みるだけで並列化の可能性を広げていると言えなくはない。しかし、実行前には判別できなかった依存関係の有無を実行中に明らかにしているだけで、もともとスレッド間に依存関係が存在するのならば、その依存関係は実行中も存在し続ける。したがって、依存関係を取り除くなど、本質的に並列性の拡大に取り組んでいるとは言えない状況であった。

(5)　　本研究代表者らも、スレッドレベル並列投機実行について従来から継続的に取り組んでいた。すでに「投機メモリ」と呼ぶ概念を考案・提唱しており、単にスレッドレベル並列投機実行を実現するという段階を超えて、本質的にスレッドレベル並列性を拡大・抽出するための検討を開始していた。

２．研究の目的

(1)　　本研究では、投機実行によって隠れた並列性を利用するだけでなく、並列性そのものの本質的な増大を図る。具体的には、スレッドレベル並列投機実行においてアボートの発生を回避することによって、スレッドレベル並列投機実行時の性能を向上させる。

(2)　　例えば、右の C 言語のコード例で、for ループのイタレーションをスレッドとして実行する場合を考える。if 文の条件が真であれば、変数 count の参照でスレッド間に依存関係が生じ、スレッドがアボートする可能性が大きい。そこで、if 文の条件が真の場合でも、プログラムのこの場所で count をインクリメントせず、スレッドのコミット時（終了時）に count のインクリメントを非投機的に行う。この例は単純であるが、count のインクリメントを含む if 文が、ループ本体から呼び出した関数の中に記述されているかもしれないし、また、単なるインクリメントではなく、より複雑な処理が記述されているかもしれない。それらの場合でも、プログラム上で記述する場所を移動（マイグレート）させるのではなく、（若干の軽微な書き換えは行うとしても）プログラムに記述する場所はそのままで、アボートの原因となる処理の実行をコミット時まで遅らせる。つまり、アボートの原因となるプログラムコードを、アボートが生じない場所に移動させてプログラムが買い換えられていたかのように実行する。この動的コードマイグレー

```
for( … ) {
   …
   if( … )
        count++;
   …
}
```

ションの仕組みを開発し、アボートの発生を回避することで、プログラムの実行性能を向上させる。

(3)　本研究の根底にあるより大きな目的は、スレッドレベル並列性抽出の機会を拡大できる可能性を学術的な見地に立って明らかにすることである。動的コードマイグレーション方式の開発は、それによって「スレッドレベル並列性の抽出機会をどこまで増やすことが可能か」という普遍的な課題に対する挑戦の過程における１ステップと言える。

３．研究の方法

(1)　まず、アプリケーションプログラムと並列投機実行システムとの間のインタフェースを決定する。具体的には、上記 2-(2)のインクリメントのような単純な処理だけでなく、どこまで複雑な処理まで動的コードマイグレーションの対象とできるかを見定める。その上で、動的コードマイグレーションによって実行を遅らせる処理を並列投機実行システムに登録する方法や、その処理で使用するデータの受け渡し方法などを検討し、並列投機実行ライブラリの機能としての仕様を設計する。

(2)　上記 3-(1)の仕様を満たすライブラリの実現方式を検討し、並列投機実行システムの概略設計を行う。

(3)　上記 3-(2)の結果に基づき、並列投機実行システムの実装を行う。

(4)　上記 3-(3)で実装した並列投機実行システムの評価環境を構築し、動的コードマイグレーション方式の性能評価を行う。

４．研究成果

(1)　動的コードマイグレーションを実現する手段として、プログラマやコンパイラが指定する任意の処理をコミット時まで遅らせるためのライブラリ仕様とその実現方式を設計し、また、実際に実装も行なった。アボートの原因となり得る処理を行うコードブロックに対して、(i)アボートの直接の原因となるデータへのアクセスと、(ii)それ以外の処理とを、できるだけ(i)の処理量が少なくなるように分離し、そのようにして分離された(i)のみを小さな関数（これを「遅延関数」と呼ぶことにする）に再構成する。投機スレッド中では(ii)の処理のみを実行し、遅延関数と(ii)の処理結果を並列投機実行システムに登録する。投機スレッドがその投機実行を終えた後、並列投機実行システムは依存関係の検査を行い、依存関係が破壊されていないことが確認できればコミット処理を行う。コミット処理の内容は、投機実行結果を正規のデータの値として確定するものであるが、ここで、本研究では、単に投機実行の結果をメモリに書き込むだけでなく、登録された遅延関数の実行を行うことにした。これにより、(i)の処理を投機的に行なっていた従来の投機実行に比べて、アボートの発生回数を削減することに成功した。

(2)　上記 4-(1)の手段は、動的コードマイグレーションを実現する基本的かつ汎用的な手段であり、コミット処理時に任意の遅延関数を実行できるようにしている。コミット処理に要する時間は並列実行によって隠蔽することができない性質のものであるので、遅延関数を小さくしたとしても、遅延関数の実行にかかる時間もやはり隠すことができない（それでも、アポートの発生回数を減らすことによる効果の方がはるかに大きい）。本研究で行なった調査・検討の結果、アプリケーションプログラムの中には、上記 4-(1)の汎用的な手段が必要となるケース以外に、より単純に動的コードマイグレーションを実現できるケースも少なくないことが判明した。そこで、そのようなケースをターゲットとして「条件付きコミット」を開発した。コミット処理では投機実行の結果をメモリに書き込むが、その書き込みを条件付きで行う（すなわち、条件が真となる場合のみに書き込みを行い、偽の場合は書き込みを行わない）ことで上記 4-(1)の手段よりも実行オーバヘッドを削減した。

(3)　従来は、投機スレッドが実行する処理の大部分でスレッド間に依存関係がなかったとしても、例えばたった 1 つのデータへのアクセスに関して依存関係が存在するだけで投機実行は失敗してしまうため、高速化は望めなかった。また、そのようなアプリケーションで純粋（非投機的）な並列処理が可能であったとしても、依存関係の原因となるデータの受け渡しのタイミングによっては同時実行可能な実際のスレッド数に制限が生じてしまい、やはり高速化は容易ではなかった。上記 4-(1)(2)の動的コードマイグレーションは、従来の並列投機実行および非投機的な並列実行のいずれと比べても、そのような依存関係による直列化の影響を非常に少な

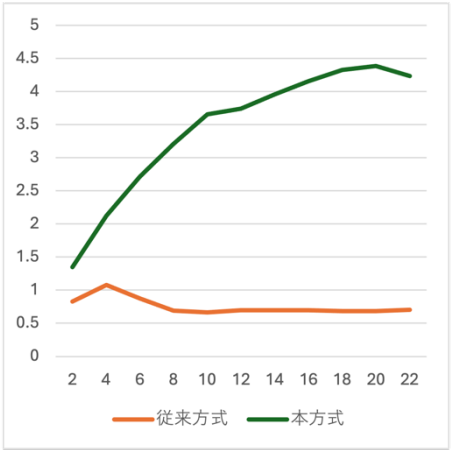
く抑えることができるため、スレッドレベル並列性の抽出機会を拡大する非常に有効な手段と言える。

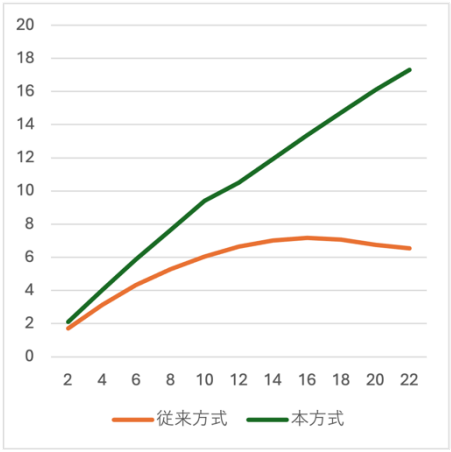
(4)　　本研究の目的は、動的コードマイグレーション技術を確立することで、スレッド間の依存関係によるアボートの発生を削減することであったが、その過程で、アボートが発生した場合にその影響を軽減するための着想を得た。つまり、アボートした場合にスレッドの実行を最初からやり直すのではなく、依存関係を破壊した時点から実行をやり直す。考え方としては古くから提案されているもので、チェックポイントリペアまたはチェックポイントリスタートと呼ばれ、すでに様々な分野で利用されている。しかし、これをスレッドレベル並列投機実行の技術分野で効率よく実装した例はまだ報告されていなかった。本研究では、チェックポイントリペアをスレッドレベル並列投機実行に導入するために、動的コードマイグレーションと同様に機能レベルから実装レベルにわたって検討を行い、効率の良い実装方式を開発するに至った。

(5)　　性能評価の結果、上記 4-(1)(2)(4)により、スレッドレベル並列化の対象範囲を拡大するとともに、プログラム実行の高速化を達成した。高速化の度合いは対象プログラムやデータサイズにもよるが、従来より大きく性能が改善されることを確認した。

①　左下図は 1 万ノードの 2 分探索木を生成する場合の性能を、単に並列投機実行した場合（従来方式）と本研究の成果を使用した場合（本方式）で比較したものである。縦軸が逐次実行（単一スレッド実行）時に対する性能向上比であり、横軸がスレッド数を表す。従来方式では並列投機実行を行なっても性能が上がらず、逆に、逐次実行時よりも遅くなっている。これに対し、本研究では、最大 4.5 倍近くまで性能を改善している。

②　右下図は 1 億ノードの 2 分探索木を生成する場合の性能を示す。ノード数が多くなるとスレッド間に依存関係が存在することも少なくなるので、従来方式でも最大 7 倍程度まで高速化を達成しているが、本研究により、さらに高速化が達成できることがわかる。





５．主な発表論文等

〔雑誌論文〕　計6件（うち査読付論文　6件／うち国際共著　0件／うちオープンアクセス　2件）

１．著者名 Hirata Hiroaki、Nunome Atsushi	４．巻 11
２．論文標題 Performance Evaluation on Parallel Speculation-Based Construction of a Binary Search Tree	５．発行年 2023年
３．雑誌名 International Journal of Networked and Distributed Computing	６．最初と最後の頁 88～111
掲載論文のDOI（デジタルオブジェクト識別子） 10.1007/s44227-023-00013-w	査読の有無 有
オープンアクセス オープンアクセスとしている（また、その予定である）	国際共著 －

１．著者名 Hirata Hiroaki、Nunome Atsushi	４．巻 1
２．論文標題 Parallel Binary Search Tree Construction Inspired by Thread-Level Speculation	５．発行年 2022年
３．雑誌名 Proceedings of the 23rd International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing	６．最初と最後の頁 74-81
掲載論文のDOI（デジタルオブジェクト識別子） 10.1109/SNPD-Summer57817.2022.00021	査読の有無 有
オープンアクセス オープンアクセスではない、又はオープンアクセスが困難	国際共著 －

１．著者名 Nunome Atsushi、Hirata Hiroaki	４．巻 1
２．論文標題 Enhancing the Performance of an Autonomous Distributed Storage System in a Large-Scale Network	５．発行年 2022年
３．雑誌名 Proceedings of the 23rd International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing	６．最初と最後の頁 87-94
掲載論文のDOI（デジタルオブジェクト識別子） 10.1109/SNPD-Summer57817.2022.00023	査読の有無 有
オープンアクセス オープンアクセスではない、又はオープンアクセスが困難	国際共著 －

１．著者名 Nunome Atsushi、Hirata Hiroaki	４．巻 10
２．論文標題 Adaptive Parameter Tuning for Constructing Storage Tiers in an Autonomous Distributed Storage System	５．発行年 2022年
３．雑誌名 International Journal of Networked and Distributed Computing	６．最初と最後の頁 1～10
掲載論文のDOI（デジタルオブジェクト識別子） 10.1007/s44227-022-00004-3	査読の有無 有
オープンアクセス オープンアクセスとしている（また、その予定である）	国際共著 －

1．著者名 Hirata Hiroaki、Nunome Atsushi	4．巻 1
2．論文標題 Reducing the Repairing Penalty on Misspeculation in Thread-Level Speculation	5．発行年 2021年
3．雑誌名 Proceedings of the 8th International Virtual Conference on Applied Computing & Information Technology	6．最初と最後の頁 39-45
掲載論文のDOI（デジタルオブジェクト識別子） 10.1145/3468081.3471120	査読の有無 有
オープンアクセス オープンアクセスではない、又はオープンアクセスが困難	国際共著 －

1．著者名 Nunome Atsushi、Hirata Hiroaki	4．巻 1
2．論文標題 An Adaptive Tiering Scheme for an Autonomous Distributed Storage System	5．発行年 2021年
3．雑誌名 Proceedings of the 8th International Virtual Conference on Applied Computing & Information Technology	6．最初と最後の頁 62-68
掲載論文のDOI（デジタルオブジェクト識別子） 10.1145/3468081.3471124	査読の有無 有
オープンアクセス オープンアクセスではない、又はオープンアクセスが困難	国際共著 －

〔学会発表〕　計4件（うち招待講演　0件／うち国際学会　4件）

1．発表者名 Hirata Hiroaki、Nunome Atsushi
2．発表標題 Parallel Binary Search Tree Construction Inspired by Thread-Level Speculation
3．学会等名 The 23rd International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing（国際学会）
4．発表年 2022年

1．発表者名 Nunome Atsushi、Hirata Hiroaki
2．発表標題 Enhancing the Performance of an Autonomous Distributed Storage System in a Large-Scale Network
3．学会等名 The 23rd International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing（国際学会）
4．発表年 2022年

１．発表者名 Hirata Hiroaki
２．発表標題 Reducing the Repairing Penalty on Misspeculation in Thread-Level Speculation
３．学会等名 The 8th International Virtual Conference on Applied Computing & Information Technology, ACIS & ACM（国際学会）
４．発表年 2021年

１．発表者名 Nunome Atsushi
２．発表標題 An Adaptive Tiering Scheme for an Autonomous Distributed Storage System
３．学会等名 The 8th International Virtual Conference on Applied Computing & Information Technology, ACIS & ACM（国際学会）
４．発表年 2021年

〔図書〕　計0件

〔産業財産権〕

〔その他〕

―

６．研究組織

	氏名 （ローマ字氏名） （研究者番号）	所属研究機関・部局・職 （機関番号）	備考
研究分担者	布目　淳 (Nunome Atsushi) (60335320)	京都工芸繊維大学・情報工学・人間科学系・准教授 (14303)	

７．科研費を使用して開催した国際研究集会

〔国際研究集会〕　計0件

８．本研究に関連して実施した国際共同研究の実施状況

共同研究相手国	相手方研究機関