Vol.2013-OS-127 No.12
Vol.2013-EMB-31 No.12
2013/12/3

メニーコアプロセッサのための高スループットな計算方法の提案

鷹野駿介　朱軼青　岡瑞起
池上高志　阿部洋丈　加藤和彦

本論文では並列計算において, 計算用のスレッドローカルな領域と計算結果をまとめる共有領域を持ったデータ構造を提案する. これによって, 並列操作をロールバックなしにほぼロックフリーで実現可能となり, 高いスループットを得られる. 提案手法を Extendible Hash に実装を行い, 実験を通して性能評価を行った.

A high throughput computational method for many-core processors

Shunsuke Takano, Yiqing Zhu, Mizuki Oka, Takashi Ikegami,
Hirotake Abe and Kazuhiko Kato

This paper proposes a novel scheme for concurrent computations on many-core processors. The scheme uses threads on local memory for computation and only accesses global memory when consolidating the computational results. This novel scheme allow us to execute concurrent process almost lock-free without roll-backs, thus achieving a high throughput. The proposed scheme is implemented using extendible hash.

1. はじめに

近年, CPU 単体のクロック数増加は頭打ちとなり, コア数を増加させることで性能の向上が行われている傾向である. そのため, 多くのコアを生かせるような並列化手法に関心が持たれている. 共有メモリ並列プロセッサにおいては, 作業を行うスレッド間でのデータ競合を防ぐ必要がある. 一般的な手法としては, データにロックを掛けることでデータの破壊から保護することができる. しかしロックを用いた手法では, デッドロックなどパフォーマンス低下の要因を防ぐためにプログラマに負担がかかってしまう.

一方で,Software Transactional Memory[1] と呼ばれるロックフリーな手法も考えられている. これは, データ操作を保存し, 検証やロールバックによってデータの一貫性を確保して楽観的に並列操作を行うものである. しかし, ロールバックは余分な計算コストとなってしまい, さらにローカルコピーが必要であるため複雑なデータ構造には不向きである. そこで, 本研究ではそれぞれのスレッド専用の領域と共有アクセスできる領域の 2 つを用意し, スレッドが自分の領域を使って独立に計算を行う状態と, 計算結果を統合する状態を交互に行うことで, ほぼロックフリーに並列計算を実現するデータ構造を提案する. 先行研究として,[2] に本手法を B-tree[3] に実装した論文を示す.

本研究では, メニーコアを持つ計算機上での並列計算において, 比較的複雑なデータ構造を利用した計算を行った時, 高いスループットを実現することを目的としている. これによって, メニーコアが一般的になった時に, そのハードウェア資源を最大限に利用できることを目指す.

2 章では提案手法に関する解説を行う. 3 章では提案手法を実装する方法を示す. 4 章では実験で性能を明らかにする. 5 章では関連研究について言及する.6 章ではまとめと今後の課題について述べる.

2. 提案手法

2.1 概　要

本手法では, Cave and Court と呼ばれるオフィスをモデルとしたデータ処理モデルを用いる. オフィスでは, 各々の労働者が作業を行う自分用の仕事場 (Cave と呼ばれる) と, 作業について話しあう会議場 (Court と呼ばれる) がある. それと同様に, 各々のスレッドが作業を行う場所として Private 領域があり, それぞれのスレッドの作業の結果をまとめた Public 領域が存在する. この領域上で, Cave mode と Court mode の 2 つの状態を使って計算が行われる.

Cave mode とは, 一般的な操作を行なっている状

Vol.2013-OS-127 No.12
Vol.2013-EMB-31 No.12
2013/12/3

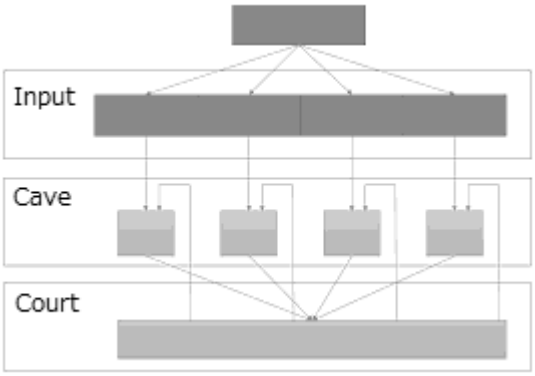

図 1 Cave and Court Computing(CCC) の流れ
Fig. 1 Overview of Cave and Court Computation

態である. これはロックフリーであり, 全てのスレッドは並列に作業を行いながら, 計算結果を自分専用の Private 領域に格納する.

Court mode とは, Cave mode によって蓄えられた計算結果を Public 領域に反映している状態である. これはロックを利用して Cave mode によるデータ書き換えを行わないようにした上で, 計算結果の統合処理が行われる. 図 1 に作業の流れを示す.

本論文では, Cave mode と Court mode を利用した計算手順を Cave and Court Computing(CCC) と定義する.

2.2 アプリケーションの例

文章に含まれる単語の数を数える word count アプリケーションを例として提案手法を適用する方法を示す. キーとして単語, 値として出現回数のペアが単位となってデータ構造に格納される.

まず, 入力が分割され, それぞれのスレッドのタスクとして課される. それぞれの単語に対してスレッドはキーと値 (単語と頻度) のペアを作成して Cave mode において, データ構造へと並列に挿入する. 特定の条件で Court mode に移行しデータ構造を再構成する. 全ての入力が終わったらさらにデータ構造を再構成することで Public 領域に全体の結果を得ることができる.

2.3 データ構造

CCC は, 様々なデータ構造に適用可能であると考えられる. 例として, B-Tree や Dynamic Hash が挙げられる.

3. 実　　装

3.1 処理の流れ

(1) 入力ファイルをスレッド数個に分割し, それぞれをスレッドのタスクをして割り当てる.
(2) Cave mode を行うことで, 入力データの処理を行う. この時はロックを用いない.
(3) Private 領域の肥大化など, 一定の条件で Court mode に移行し, その後 Cave mode に戻る.
(4) 全ての入力の処理が終わるまで繰り返し、最後に Court mode を行うことで, 全ての計算結果を Public 領域に反映させる.

3.2 入 力 処 理

入力には領域を 2 つ用意し, 一方は Cave mode で処理を行う際, 作業用スレッドが参照するための領域, もう一方は入力を保存するための領域とする. このダブルバッファリングによって効率的に入力を処理することができる.

3.3 データ構造

1 ページには, 予め定められた大きさの Public 領域と, スレッド数個の Private 領域が確保されている. それぞれの Private 領域の大きさは特に定まっていない.

3.4 Cave mode

Cave mode で行う操作は search, insert, delete の 3 つである. これらについて説明する.

3.4.1 search

search 関数では, 検索したい値をキーとして受け取り, そのキーに対応した情報を返却する. CCC において, 検索を行うスレッドはキーが含まれる部分に到達した後, Public 領域と自分の Private 領域のみ参照可能であるため, 自分の領域以外の Private 領域にデータが存在する場合, データに inconsistency が起こることがある. データ構造全体を Court mode によって再構成をすることで, inconsintency を解消することができる.

3.4.2 insert

insert では, キーとして単語, 値として出現回数の 1 のペアが与えられる. まずキーにハッシュ関数を適用し, それによって示されるページにアクセスする. 次に保存に用いる領域を確保し, 自分の Private 領域に格納する.

3.4.3 delete

delete 関数では, キーを入力として受け取り, そのキーの情報をデータ構造から削除する. CCC にといては, この関数が割り当てられたスレッドの Private 領域に削除情報を一時的に格納される.

Vol.2013-OS-127 No.12
Vol.2013-EMB-31 No.12
2013/12/3

3.5 Court mode

Court mode で行う作業である Reorganization について説明する.

3.5.1 Reorganization

この関数は CCC のための特別なものとなっている. まず, 再構成する領域を決定し, データ構造にロックを掛ける. そして, 作業を行うスレッドが全ての Private 領域のデータと Public 領域のデータを統合し, 最新のデータとして Public 領域に格納する. この時, Public 領域のデータ数が大きく増加するため, 実装されたデータ構造に応じた処理を同時に行うことがある.

この関数を実行するためには, 全てのスレッドが Cave mode の作業を終了している必要がある. そのため, 再構築が要請された時, 全体からアクセスできるフラグを立てる. それぞれのスレッドが Cave mode の作業を終えた時, このフラグをチェックし, 立っていれば次の作業を行わず待機する. そうして全てのスレッドが待機していることを確認できた後に, 実行可能となる.

3.6 mode の切り替え

Cave mode と Court mode の切り替えは様々な条件をトリガーとして行うことができる. 例として, Private 領域の大きさに制限を設けて, 超過した時や一定のサイズのデータを処理した後などがある.

3.7 データの出力

全てのスレッドの作業が終了の後, データ構造全体を再構成する. これによってデータの一貫性が保証されるので, public 領域に保存された計算結果を順に出力することで, アプリケーションの正しい結果を得ることができる.

4. 実　　験

CCC は B-Tree や Dynamic Hash など, 一般的なデータ構造に実装可能であるが, ここでは, Dynamic Hash の 1 つであるを Extendible Hash[4] に実装した.

今回はアプリケーションとして word count を実装した. これは, 入力データに含まれるそれぞれの語ごとに, その出現回数を数え上げるものである.

mode を切り替えるトリガーとして, Private 領域のサイズが一定を超えた場合と, 入力を分割してその 1 セットが終了した時としている.

追加機能として, フィードバック機能を実装した. 入力を行う分割サイズを分割された入力に対する処理が完了するまでのスループットを比較して, 最適な値を探索している.

具体的には, 一定の幅で入力サイズを変動させてそれぞれスループットを計測する. そして, その状態で最良の入力サイズを選択する.

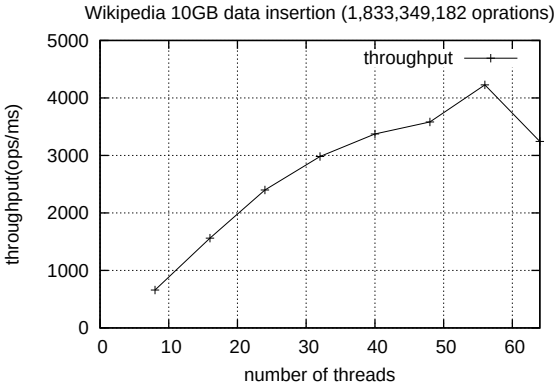

図 2 word count でのスループット
Fig. 2 throughput on word count

4.1 実 験 環 境

実験環境として, CPU が AMD Opetron(TM) Processor 6272 2.1Ghz 8Cores(x2x4), OS が, CentOS release 6.3(Final), メモリが 64GB, 開発言語が, C++11, コンパイラとして GNU GCC 4.7.2 を用いている. この CPU は Non-Uniform Memory Access(NUMA) アーキテクチャである.

4.2 実験データ

入力データとして, Wikipedia 英語版を 10GB 分利用して word count を行い, スループットを計測する. 通常の実装についての評価と, フィードバック機能の評価の 2 種類を行う.

4.3 実 験 結 果

図 2 に実験結果を示す. スレッド数 56 まではスループットが改善されていることがわかる.

図 3 に各 mode の計算時間を示す. Extendible Hash では, 要素へのアクセスが O(1) であるため Cave mode の時間が短くなっている.

図 4 に通常実行とフィードバック機能を追加したものとを比較した結果を示す. フィードバックの導入により, 適切な I/O サイズが選択され, スループットが改善していることがわかる.

5. 関 連 研 究

データを並列に処理を行うという点では, 分散コンピューティングが存在し, MapReduce[5] などが開発され, 利用されている. MapReduce は, クラスタ上での分散処理を行うことで, ビッグデータ処理に対して高いスループットの実現している. 本研究では, メニーコア向けの並列計算に着目している. スケーラビリ

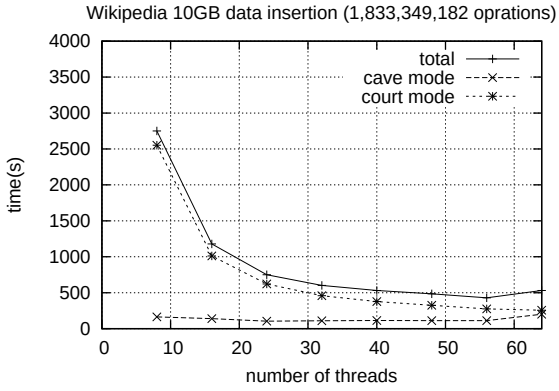

図 3 各モードの計算時間
Fig. 3 computational time of cave mode, court mode, and the total

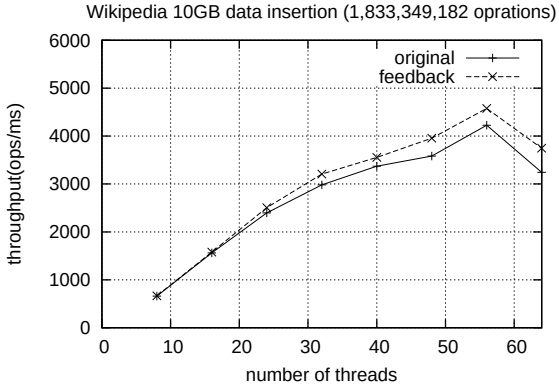

図 4 フィードバックによるスループット
Fig. 4 Throughput with feedback mechanism

ティはコア数の制約を受けるが, これからメニーコアの一般化が期待される中で, コスト面での削減が期待できる.

6. 結　　論

メニーコア環境における並列計算方式, Cave and Court Computing を提案した. 本手法を適用することで, 一般的なロックアルゴリズムにて生じるスレッドブロックによる並列度低下を解消し, さらにデータの複製を行うことなく並列計算を行うことで, スループットの向上が確認できた.

今後の課題として, CCC には, Court mode に移行するタイミング等複数のパラメータがあるため, 最適化を行うことでより高いスループットを実現できるようにしたい. また, 計算が複雑になるアプリケーションの実装を行い本手法が有効であるデータサイズや計算の複雑度について明らかにしていきたい. さらに, 実装するデータ構造によってもデータアクセス速度や格納速度に差が生じるため, アプリケーションによって適切なデータ構造を調べていきたい.

参 考 文 献

1) Shavit, N. and Touitou, D.: Software transactional memory, *Proceedings of the fourteenth annual ACM symposium on Principles of distributed computing*, PODC '95, New York, NY, USA, ACM, pp. 204–213 (1995).
2) 朱 軼青, 鷹野 駿介, 陳 ウェイ, 岡 瑞起, 加藤和彦:A High Throughput Computing Scheme on Many-core Processors, 日本ソフトウェア科学会第 30 回大会セッション, September, Tokyo, 2013.
3) Bayer, R. and McCreight, E.: Organization and maintenance of large ordered indices, *Proceedings of the 1970 ACM SIGFIDET (now SIGMOD) Workshop on Data Description, Access and Control*, SIGFIDET '70, New York, NY, USA, ACM, pp. 107–141 (1970).
4) Fagin, R., Nievergelt, J., Pippenger, N. and Strong, H. R.: Extendible hashing-a fast access method for dynamic files, *ACM Trans. Database Syst.*, Vol.4, No.3, pp.315–344 (1979).
5) Dean, J. and Ghemawat, S.: MapReduce: simplified data processing on large clusters, *Commun. ACM*, Vol. 51, No. 1, pp. 107–113 (2008).