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

平成　２９　年　　６　月　１２　日現在

機関番号：　１２６０８
研究種目：　挑戦的萌芽研究
研究期間：　2014～2016
課題番号：　２６５４００４２

研究課題名（和文）ハードウェアトランザクショナルメモリと従来の排他制御を組合せたデータ構造の設計

研究課題名（英文）Designing concurrent data structures: combining hardware transactional memory and traditional mutual exclusions

研究代表者

宮崎　純（Miyazaki, Jun）

東京工業大学・情報理工学院・教授

研究者番号：４０２９３３９４

交付決定額（研究期間全体）：（直接経費）　　2,900,000円

研究成果の概要（和文）：近年、マルチコアCPUやGPUといったメニーコアプロセッサにより計算の高速化を目指す研究が活発である。しかし、共有メモリ型プロセッサにおいて、多くのアプリケーションではデータ構造を共有するため、その排他制御が重大な問題となっている。
この問題の解決のため、最近利用可能となったハードウェアトランザクショナルメモリを利用して、並行データ構造を構成する方法について、LRUやB-treeを事例に研究を行った。さらに排他制御が利用できないGPUを利用した高並列処理を目指して、新たに提案した辞書プリミティブと既知のデータ並列プリミティブとを組み合わせて、一般的なテキスト処理が大幅に高速化できることを示した。

研究成果の概要（英文）：Recently, researchers have actively studied on designing faster algorithms and their data structures for manycore processors, such as a multicore CPU and a GPU. Since data structures in many applications are shared by concurrent threads in these shared memory processors, the mutual exclusion for them becomes one of the serious bottlenecks.
To cope with this problem, this study presents on configuring concurrent data structures using hardware transactional memory which is recently available through case studies on LRU and B-tree. In addition, aiming at highly parallel processing on a GPU which can hardly use mutual exclusion, it is revealed that realistic text processing can greatly be accelerated by combining the dictionary primitive that we proposed and existing data parallel primitives.

研究分野：　データ工学

キーワード：　並行データ構造　ハードウェアトランザクショナルメモリ　排他制御　データ並列プリミティブ　性能評価

１．研究開始当初の背景

近年、メニーコアプロセッサにより計算処理の高速化を目指す研究が活発である。しかし、このような共有メモリ型プロセッサにおいて、多くのアプリケーションではデータ構造等を共有するため、その排他制御が処理を阻害する要因となっている。従来のロック等の排他制御はオーバヘッドが非常に大きいため、その解決案としてロックを排除したロックフリーあるいはウェイトフリーアルゴリズム(以降単にロックフリーアルゴリズムと省略)が提案されてきた。ロックフリーアルゴリズムは正しく動作させるための設計が極めて難しいため、単方向リストやハッシュ等の簡単なデータ構造しか実現されていないが、過去のロックフリーバッファ管理の研究成果からも多数の書込み競合に強いという利点が判明している。

一方、ハードウェアトランザクショナルメモリ(以降 HTM と省略)と呼ばれるハードウェアで複数のメモリ操作をアトミックに実行可能な仕組みが提案され、容易に利用可能となった。しかし、HTM は多数の同時書込みが発生すれば、競合あるいはハードウェアリソースの枯渇によりロールバック処理が頻発し、オーバヘッドが大きいことが知られている。

さらに、GPU 等のコアの機能を大幅に限定したメニーコアプロセッサでは、排他制御のためのアトミック命令は存在してもオーバヘッドが極めて大きく、実用的な使用は困難である。そのため排他制御そのものを排除した計算処理が必要となってきている。

２．研究の目的

メニーコアプロセッサ時代において、プログラマに特殊技術を強いることなく高性能のソフトウェアを作成するためには、多数の同時書込みが発生する複雑なデータ構造であっても、正しい動作と低オーバヘッドを実現しなければならない。

本研究では、メニーコアプロセッサを利用するソフトウェアにおいて、多数の同時書込みが発生する複雑なデータ構造であっても、低オーバヘッドで排他制御され、かつ正しく動作させるための方法を明らかにする。具体的には、ロックフリーアルゴリズム、HTM、ロックの併用により、書込み競合が主体の部分と読出し競合が主体の部分、競合があまり発生しない部分とに分け、これら三種類の排他制御法の利点を活かせるよう適材適所に使用してデータ構造を構成する手法の研究を行う。また、評価実験を通して提案手法の有効性の検証を行う。

さらに、GPU 等のメニーコアプロセッサにおいては排他制御を行わずに実用的な並列処理を行うためのソフトウェア構成方法の事例研究を行う。

高性能計算のための近年の大きなアプローチであるメニーコアプロセッサの利用において、共有データ構造の排他制御に付随する大きなオーバヘッドの問題の解決が必要である。これを解決することで、単なる技術的な問題だけでなく、ソフトウェアの設計を容易にし、高性能なソフトウェアの生産性の向上に貢献することを目的とする。

３．研究の方法

本研究は、メニーコアプロセッサ上での高性能計算を可能とするため、共有データ構造へのアクセスのオーバヘッドの低減を目標としているが、問題点の把握を容易にするため、次に挙げる三つの研究項目を中心に研究を進める。

- 複雑なデータ構造において、排他制御箇所のデータの書込みと読出しのスペクトラム、競合の頻度に応じて、ロックフリーアルゴリズム、HTM による排他制御、ロックを適材適所に使い分ける手法の検討
- 三種類の排他制御法を組み合わせた時、特に HTM による排他制御とロックの関係が性能に及ぼす影響の解析と、その解決方法の検討
- 排他制御のない高並列計算処理の設計のためのプリミティブの組み合わせ法

三種類の排他制御法のうち、ロックフリーアルゴリズムが最も低オーバヘッドであることは明らかであるが、プロセッサアーキテクチャに依存しない既知のロックフリーアルゴリズムは、単一方向リスト、スキップリスト、ハッシュ等の基本的なもののみである。しかし、トライや二分探索木、平衡木等の木構造やグラフに関連するデータ構造など、効率の良いデータの探索や更新が必要とされるときによく使用されるデータ構造に関しては、ロックフリーアルゴリズムは知られていない。まず、ケーススタディとして、ロックフリーアルゴリズムが知られていないが頻繁に利用される LRU のデータ構造について、三種類の排他制御法を組み合わせて低い排他制御オーバヘッドのデータ構造の設計を試みる。このケーススタディを通して、そのデータ構造に対する探索や更新時に、どのような挙動を示すのかを解析することを試みる。また、解析ツールを利用して、実行時の情報を取得し、実際にどの箇所が競合するのかを調査する。この結果に基づいて、データ構造の各部分に適した排他制御方式の選択方法について明らかにする。

引き続いて、大規模データ管理で一般的に使用される B-tree について、そのスループット性能向上のために、HTM と既存の排他制御法とを組み合わせる手法を中心に研究を行う。

また、近年の高性能並列計算で必須の GPU に代表されるメニーコアプロセッサ上で、非

数値演算処理に対して排他制御を行わないようにするためのソフトウェアの設計方法の検討を行い、テキスト処理を事例としてデータ並列プリミティブの組み合わせ法や新たなプリミティブの研究を行う。

４．研究成果

はじめにケーススタディとして、LRU データ構造について、単方向リストとハッシュ表を組み合わせて構成し、IBM の Power8 プロセッサ(12 コア/3.026GHz) を利用して比較実験を行った。LRU はページ置換アルゴリズムの一つであり、OS やデータベースなどで広く使われている。LRU は一般的に双方向リストとハッシュ表によって実装されるが、本課題ではデッドロックの問題を回避するために、単方向リストを用いた。

単方向リストとハッシュ表ともにロックフリーアルゴリズムが知られており、これに加えてロック、HTM による実装が可能である。すなわち LRU データ構造の構成には 9 通りの組み合わせ方法がある。

リストについては、スレッド数が多く read 主体の場合は HTM による実装、それ以外はロックフリーアルゴリズムの性能が良かった。ハッシュについては、いずれの実装も同程度であるが、read 主体の場合は HTM によるものが最も悪い結果となった。

以上を踏まえて、9 通りの組み合わせで実装した LRU に加え、LRU 全体を粗粒度でロックし操作を行う Coarse-grain Lock LRU の計 10 種類の LRU 構造に対して実験を行った。

スレッド数が少ない場合を想定した 4 スレッドでの実験では、LRU 全体をロックする Coarse-grain LRU を除き、単方向リストとハッシュ表の両方をロックフリーアルゴリズムで実装した LRU が最も良い性能であった。これは、LRU エントリの範囲が大きくなっても変わらない。また、個別の単方向リストやハッシュ表の実験結果から得られた特徴も、それらを組み合わせた LRU において確認することができた。例えば、ハッシュ表をロックで実装したものに固定し、単方向リストの排他制御法を変えた場合、単方向リスト単体の性能の順が LRU の性能の順序と一致した。

加えてスレッド数が多い場合を想定した 24 スレッドの実験も行ったが、概ね 4 スレッド時と同様であった。ただし、ハッシュ表にロックフリーアルゴリズムを使用した場合は、単方向リストに HTM を利用する LRU は極端に性能が悪くなることが判明した。これはハッシュ表のスループットが上がり、単方向リストでスレッド間の衝突が発生しやすくなったからと考えられる。一方、粗粒度ロックを利用する方法は、アクセスが集中する場合の性能は悪いが、アクセスが分散していれば、双方ともロックフリーアルゴリズムを利用する LRU を除いて、性能が良いことが明らかとなった。これはトラフィックが多いときに発生するアトミック命令のオーバヘッドや HTM のアボートによる性能低下よりも、逐次実行の方が全体として排他制御に関連するオーバヘッドが少ないということを示しており、興味深い結果と言える。

次のケーススタディとして、大規模データ管理に利用される B-tree 構造について、異なる排他制御の組み合わせによる性能向上を試みた。B-tree の完全なロックフリーアルゴリズムは知られておらず、本課題ではロックと HTM の組み合わせについて考察した。ベースとなる B-tree の構成は、Blink-tree と OLFIT を利用した。Blink-tree は内部ノードにサイドポインタと、そのノードを根とする部分木中の最大のキーを保持することにより、探索時に高々1 ノードのロックのみで排他制御を行う方法であり、スプリット時の不完全な木構造でも探索処理が可能となっている。一方、OLFIT は、ロックの実装を工夫し、探索時にはロックを取らずにノードの読出し前と後のそのノードのバージョン番号が一致していることを確認することで、一貫したアクセスが可能である。そのためアトミック命令によるオーバヘッドはない。しかし、データの更新時には従来のロックと同様にアトミック命令が必要であり、ロック開放とともにバージョン番号を上げることで探索処理の読み出しと更新処理間の一貫性を保っている。OLFIT は Blink-tree 構造に適用することが可能であり、本課題では簡単のため、Blink-tree のデータ構造に OLFIT の排他制御を適用したものを OLFIT と呼ぶことにする。

まず Blink-tree と OLFIT について、それぞれロックと HTM での実装の性能比較を行った。その結果 read 率が高いアクセスは Blink-tree を HTM 化したもの、write 率の高いアクセスは OLFIT そのものの性能が高いことが分かった。

この結果をもとに、OLFIT に対して HTM を部分的に適用した二つの手法を提案した。一つ目の手法は内部ノードに対する排他制御を HTM、葉ノードを OLFIT とする手法である。これは、アクセスが read 主体である場合を想定したものであり、OLFIT は探索時においてノードのアクセスによるオーバヘッドがあるため内部ノードでは HTM が有利に働くと考えられる。この手法を OLFIT-leaf と呼ぶ。二つ目の手法は内部ノードに対する排他制御を OLFIT、葉ノードを HTM とする手法である。これは、アクセスに write が発生する場合を想定し、葉ノード周辺のスプリット処理は競合が少ないとの仮定で、HTM で楽観的にスプリット処理を行うものである。この手法を HTM-leaf と呼ぶ。

評価実験には、Intel Xeon E5-2650v4 プロセッサ(12 コア/2.20GHz)を使用した。

read 率 100％の場合は、予想に反して OLFIT-leaf の性能が悪く、HTM の利点を活かせないことが判明した。原因はキャッシュの

容量性ミスによるアボートが多いことによる。read 率が高い場合には、OLFIT を HTM 化した手法が最も良いとの結論が得られた。

write 比率が上がると、予想通り HTM-leaf が最も性能が良いことが分かった。write 率が特に高い場合には、HTM-leaf の性能の高さが顕著となる。OLFIT-leaf では内部ノードでのトランザクションアボートが起こったが、HTM-leaf は内部ノードの処理が確実に行われるためと考えられる。全体の結果から、概ね write の比率が 10%以上であれば、HTM-leaf が有効であることを明らかにした。

本課題ではさらに研究を発展させ、マルチコアプロセッサやメニーコアプロセッサでの計算性能を低下させる排他制御を排除した並列処理についても研究を行った。特にコア数が 1000 以上からなる GPU での処理は、メモリアクセスのスループットを最大化することが重要であり、排他制御を前提とすることはできない。すなわち、一般的な並行データ構造やアルゴリズムを利用することは難しい。現在のところ、GPU を利用した計算の高速化は数値演算に代表される規則的な処理が主体となっている。しかしながら、テキスト処理に代表される不規則なメモリアクセスや可変長のデータを処理する応用は、情報検索をはじめとして多い。そのためのツールとして、データ並列プリミティブと呼ばれ、機能は単純であるが GPU で排他制御を行うことなく高い並列演算を行うアルゴリズムがいくつか提案されている。例えば、scan、sort、 compact、 load-balancing search、count といったデータ並列プリミティブが代表的である。

本課題では、高い並列性が要求される不規則データアクセスの応用例として、文書検索に利用される BM25 の計算を取り上げ、その効率的な計算手法について検討した。ボトルネックになりうる文字列操作について、有用なデータ並列プリミティブとして辞書を提案し、語を整数値 ID に変換することで可変長データに関連するボトルネックを回避しつつ、既知の排他制御を必要としないデータ並列プリミティブとを組み合わせて BM25 の計算を行う手法の提案を行った。

具体的には、compact を利用した単語の切り出し、scan による文書 ID 割当て、count による文書長計算、sort による単語のグルーピング、count と load-balancing search による大域的統計量の計算等により BM25 の骨格部分の計算を行う。これに加えて高並列処理可能な簡潔データ構造を利用した辞書で入力文書から単語を整数化するプリミティブにより、処理時間の大部分を占める sort のコストを下げることが可能となる。

TREC ClueWeb09 Category B に含まれる英文 Web 文書の単語の出現頻度の情報を利用した人工的な文書を用いて評価を行った。比較実験では、マルチコア CPU 上での MapReduce フレームワークを利用する方法(MRP)、GPU 上の MapReduce フレームワークを利用する方法(MRM)、提案手法で辞書プリミティブを利用しない方法(PP)、提案手法で辞書プリミティブを利用して最適化する方法(PPD)を用いた。実験には、CPU に Intel Core i7-6700K(4 コア/4.0GHz)、 GPU には NVIDIA GeForce GTX 970 (1664 コア/1.05GHz)を用いた。

実験結果から CPU による MRP と比べて、提案手法であるデータ並列プリミティブを用いた PP、PPD が高速となる結果となった。特に辞書プリミティブを用いた PPD では、マルチコア CPU による MRP 手法と比べて、5 倍以上の性能向上となった。辞書を用いる PPD は、PP と比べて 3.8 から 4.5 倍の性能向上を達成することができた。一方で、GPU の MapReduce による MRM は、CPU の MapReduce(MRP)よりも実行時間が長い結果となった。これは、各スレッドでの負荷の偏りが原因と考えられる。

提案手法であるデータ並列プリミティブを用いた手法について、PP では文字列のまま語を扱うため、不規則な長さの語を扱う必要があるステップでソートや大域的統計量の計算が支配的となった。それに対して PPD は辞書プリミティブを用いて語を整数に変換するステップを設けることで、辞書処理のオーバヘッドが加わるものの、その影響は小さく、その後の計算において文字列の代わりに整数値を用いることでき、以降の計算処理の実行時間が大幅に削減できた。これにより、例えば約 12 万個の Web 文書を 0.4 秒程度で処理可能となった。

以上から、GPU では高コストとなる排他制御を排除するために、データ並列プリミティブに高並列処理可能な辞書プリミティブを組み合わせることで、テキスト処理という不規則メモリアクセスと可変長データ処理を含む現実の応用に対して高い効果があることを示した。

５．主な発表論文等

（研究代表者、研究分担者及び連携研究者には下線）

〔雑誌論文〕（計 1 件）

(1) Toshiaki Wakatsuki, Atsushi Keyaki, Jun Miyazaki, “A Case for Term Weighting using a Dictionary on GPUs”, Proceedings of 28th International Conference on Database and Expert Systems Applications, LNCS, Springer, 査読有, Aug., 2017. (to appear)

〔学会発表〕（計 4 件）

(1) 蔡弘聖, 宮崎純, “HTM を利用した並行 B-tree の一手法”, 電子情報通信学会コンピュータシステム研究会 (CPSY), 電子情報通信学会技術研究報告, Vol. 117, No. 44, pp. 33-38, 北海道登別市,

May. 2017.
(2) 若月駿尭, 欅惇志, 宮崎純, “GPU 向け簡潔データ構造による辞書を利用した効率的な語の重み計算”, 情報処理学会 DBS 研究会, 東京, Jan. 2017.
(3) 永井光, 宮崎純, “複数の排他制御法を利用した高性能データ構造の構成について”, 電子情報通信学会コンピュータシステム研究会 (CPSY), 電子情報通信学会技術研究報告, Vol. 116, No. 19, pp. 27-32, 富山県黒部市, May. 2016.
(4) 若月駿尭, 欅惇志, 宮崎純, “効率的なテキスト処理を目指した簡潔データ構造を用いるトライ木 の GPU 上での実装”, 第8回データ工学と情報マネジメントに関するフォーラム(DEIM2016), DEIM2016 論文集, 福岡県福岡市, Feb. 2016.

〔図書〕(計０件)

〔産業財産権〕

○出願状況 (計０件)

○取得状況 (計０件)

〔その他〕
ホームページ等

６．研究組織
(1)研究代表者
宮崎　純 (MIYAZAKI, Jun)
東京工業大学・情報理工学院・教授
研究者番号：40293394