2K-8

HTMを活用した投機的ヘルパースレッドによる高速化の検討

松野　穣 † 大津　金光 † 大川　猛 † 横田　隆史 †
† 宇都宮大学大学院工学研究科情報システム科学専攻

1　はじめに

近年, マルチコアプロセッサによる並列処理の研究が盛んに行われている. その中で, 従来の並列化手法では高速化が困難な本質的に逐次な処理に対しても, マルチコアプロセッサを有効に活用した高速化手法の開発が求められている. その手法として投機的実行方式が有望であり, 主に実行を進めるメインスレッドとそれを支援するヘルパースレッドを用いる手法が存在する. またインテル社の最新プロセッサには並列処理で有用な機能としてハードウェアトランザクショナルメモリ (HTM) である TSX[1] が搭載されており, 今後 HTM が容易に利用できる環境の普及が予想される.

そこで本稿ではインテル社の最新プロセッサに搭載されたこの HTM を活用したヘルパースレッドによる高速化手法を検討する.

2　トランザクショナルメモリ (TM)

TM は並列処理の各スレッドの共有変数へのアクセスの仕組みであり, データの一貫性の保持に使われる. 従来, 共有変数へのアクセスは排他制御を必要とし, 競合するスレッドがなくても逐次なアクセスを行っていた. それに対して TM は並列処理を進める上で競合が起こらないと楽観的に共有変数へのアクセスを行うため, 高速なメモリアクセスを可能とする. もし競合が発生した場合, スレッドはそれまでの実行内容を破棄し, 処理を再試行することでプログラム結果の正当性を保証する. 各スレッドが待機せずに処理を進めることで排他制御に伴うオーバーヘッドがなく, 競合頻度の低いプログラムでは高い性能向上が見込まれる.

TM にはソフトウェアによる実装 (STM) とハードウェアによる実装 (HTM) がある. 基本的により高性能な HTM をサポートするプロセッサは一般的ではなかったが, インテル社の最新プロセッサが HTM に対応したことにより利用可能環境の増大が予想される.

3　ヘルパースレッド

ヘルパースレッド手法はメインスレッドの実行と並行して他のプロセッサコア上でヘルパースレッドを起動し, メインスレッドが後で実行する可能性の高い処理を先に行う手法である. 図 1 のようにメインスレッドがヘルパースレッドによって事前に処理された結果を利用することで処理時間の短縮を図る. このとき, ヘルパースレッドは投機的に処理を行うため, メインスレッドとの依存関係により誤った処理が起こり得る. その場合, ヘルパースレッドの処理結果は破棄される.

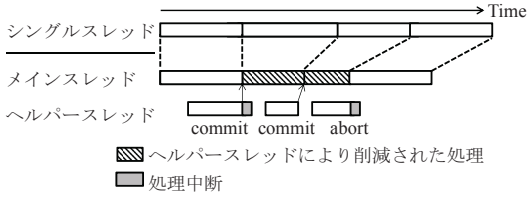

図 1: ヘルパースレッドによる高速化

4　HTM を利用したヘルパースレッド

前節で述べたようにヘルパースレッドは投機的に処理を進めるため, 誤った処理の可能性がある. それにより投機処理が全て完了するまで結果を全体に反映させてはならず, 完了時に競合の発生が確認されたなら結果を破棄し, ヘルパースレッドは改めて将来実行され得る処理を始める必要がある. 本稿ではこれらの動作の実現に HTM の活用が有効だと考え, HTM を活用したヘルパースレッドによる高速化の評価を行う.

この検討を目的として, グラフ理論における最短経路問題を解くダイクストラ法のプログラムを実装した. これは文献 [2] にて行われたものだが, その評価はシミュレーションによるもので, 想定するプロセッサ構成が現実では異なる. そこで我々はインテル社の最新プロセッサに搭載された HTM を用いて再評価を行う.

ダイクストラ法は開始点から, それに最も近い頂点へと経路を徐々に延ばしていき, 終了点に達するまでこれをくり返すことで最短経路を求める手法である. 最近の経路を高速に求めるためにバイナリヒープを用いて実装する. このプログラムにおいてヘルパースレッドはメインスレッドが将来到達する頂点を先行して処理する. ヘルパースレッド処理の共有変数へのアクセスは HTM で監視し, メインスレッドとの競合が発生した場合に備える. これによりヘルパースレッドは処理完了時までに競合がない場合のみ反映され, 競合が存在した場合は結果を破棄して再度実行をやり直す.

5　評価実験

上述のダイクストラ法を複数の入力データ, スレッド数で実行した. 評価に使用した PC の環境を表 1 に示す. またプログラムへの入力として使用するグラフデータは Recursive Matrix グラフモデル [3] で生成したものを使う. グラフデータの頂点数は 1 万, 2 万, 5 万の 3 種を使用し, 辺の数は 1000 万と 1 億, 辺の重みの範囲は 1 から 100 と 1 から 5000 の 2 種類を用いる.

Consideration of Speedup by Speculative Helper Thread Utilizing Hardware Transactional Memory
†Yutaka Matsuno, Kanemitsu Ootsu, Takeshi Ohkawa and Takashi Yokota
Department of Information Systems Science, Graduate School of Engineering, Utsunomiya University (†)

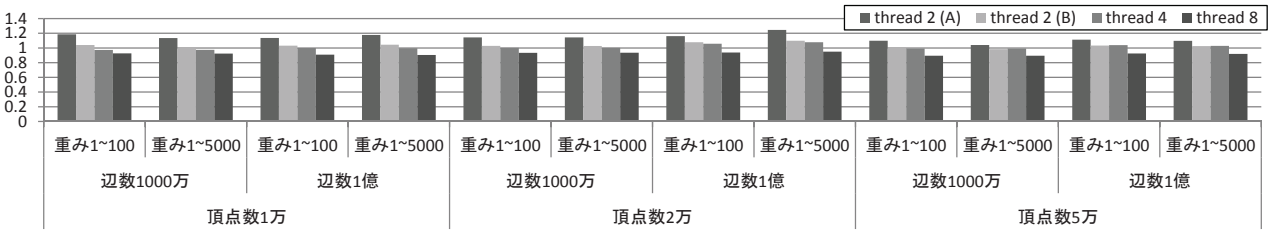

図 2: 逐次実行に対する速度向上率

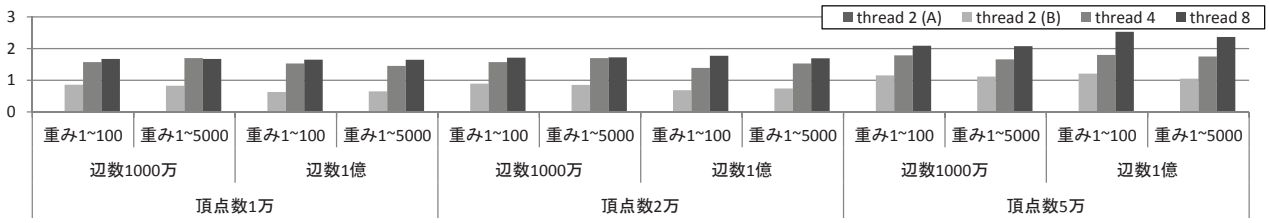

図 3: 図 2 における 1 トランザクション当たりの平均アボート回数

これらの各条件でプログラムを 50 回実行して, その平均を取った速度向上率を図 2 に, 1 トランザクション実行当たりのアボート回数を図 3 に示す. 2 スレッド実行ではほとんどアボートしない場合とほとんどアボートする場合に二極化していたため, 各々を thread 2 (A), thread 2 (B) として分けて示す. 図中の thread 2, thread 4, thread 8 はメインスレッドが 1 つとヘルパースレッドが 1 つ, 3 つ, 7 つあることを意味する.

結果として thread 2 (A) 以外は逐次実行同様かそれ以下の性能を示した. ヘルパースレッドの平均アボート回数は thread 2 (A) 以外, 高い値を示している. 文献 [2] では頂点数 1 万, 辺数 100 万の 4 スレッド実行で 1.6 倍の速度向上率と 0.003 回の平均アボート回数が報告されている. 辺数の増加に反比例したアボート回数も報告される中, 本稿の評価条件はそれ以上の辺数で高いアボート回数を示している. またヘルパースレッドの処理を反映した結果, 後からメインスレッドによる再計算が発生し, ヘルパースレッドの処理が高速化に寄与しない事例があった. これらがヘルパースレッド実行による高速化の阻害要因だと推測される.

今回のプログラムにおいて, ヘルパースレッドがメインスレッドに対して離れている場合は依存違反が起こりやすくなる. 一方, 逆に近い場合はメインスレッドの処理がすぐに追いつくため, 両者の間を一定以上離す必要がある. しかし, プロセッサコアの処理性能が高い場合は両者の距離をより遠ざける必要があるため, ヘルパースレッドによる性能向上が難しい. 文献 [2] で使用されたプロセッサは本稿で使用したプロセッサに対して処理性能が著しく低いため, このような状況が起こらず性能向上できたものと考えられる.

表 1: 評価環境

CPU	Intel Core i7 4770 3.40GHz 4 コア/8 スレッド
L1 cache	命令 32KB/コア データ 32KB/コア
L2 cache	256KB/コア
L3 cache	8MB (コア間共有)
Memory	32GB
OS	CentOS 6.5

6 おわりに

本稿では, HTM を活用したヘルパースレッドによる高速化を検討して実験を行った結果, 2 スレッド実行においては性能向上した例を確認したものの, その他の条件においては有意な性能向上を確認できなかった. この原因は, 評価環境のプロセッサコアが高性能であることによりヘルパースレッドが有効に機能する状況になりにくかったためだと考えられる. 2 スレッド実行時のトランザクションの平均アボート回数の二極化の原因もまだ判明しておらず, 今後はそれらの原因究明と対処法の検討を行う予定である.

謝辞

本研究は，一部日本学術振興会科学研究費補助金 (基盤研究 (C)24500055，同 (C)24500054, 同 (C)25330055, 若手研究 (B)25730026) の援助による．

参考文献

[1] Intel: “Intel64 and IA-32 Architectures Software Developer’s Manual Volume 1: Basic Architecture”, chapter 15. 2013.

[2] Konstantinos Nikas, Nikos Anastopoulos, Georgios Goumas, Nectarios Koziris: “Employing Transactional Memory and Helper Threads to Speedup Dijkstra’s Algorithm”, Proc. of 2009 International Conference on Parallel Processing, pp.388-395, 2009.

[3] D. Chakrabarti, Y. Zhan, and C. Faloutsos: “R-mat: A recursive model for graph mining”, Proc. of 4th SIAM International Conference on Data Mining, 2004.