計算システム工学分野

題　　目 プログラム区間ごとの特徴を考慮した効率的なスレッド統合手法

学籍番号 23413541 氏名　祖父江　宏祐

指導教員名　津邑　公暁

1 はじめに

マルチコア・プロセッサのリソースを有効利用するための方法の一つとして，プログラムのマルチスレッド化がある．マルチスレッドプログラムにおいて，生成されるスレッドの数はプログラム実行効率に大きな影響を与えるため，適切なスレッド数を自動的に決定するような機構の必要性は高い．これを実現する機構として，Thread Tailor[1] が提案されている．Thread Tailorは，プログラムを実行するプロセッサに適切なスレッド数を判断し，各プロセッサコアの処理量が均衡化されるようにスレッドを統合する．しかしながら，Thread Tailorには2 つの問題が存在する．1つ目の問題として，既存のThread Tailorはプログラム開始時に1 度しかスレッドを統合しないため，プログラムの負荷の変動に対応できないことが挙げられる．2つ目の問題として，プログラム実行時にデッドロック発生の恐れがあることが挙げられる．本研究では，これらの問題を解決する手法を提案する．

2 Thread Tailor

Thread Tailor では実行するスレッド数，およびどのスレッド同士を統合するかを決定するために，まずプロセッサの使用可能リソースを調査する．調査する項目は，プロセッサのコア数，2 次キャッシュ容量，メモリバンド幅容量である．次に，各スレッドの挙動を事前にプロファイリングする．この際に収集するデータは，並列化対象関数を実行した際の各スレッドの実行サイクル数，ワーキングセットサイズ，使用メモリバンド幅，そしてスレッド間の通信コストである．そして，プロファイリングによって得られた結果からコミュニケーショングラフを生成する．このグラフの各ノードは，スレッドの実行サイクル数，メモリバンド幅，ワーキングセットサイズを表し，ノード間のエッジはスレッド間の通信コストを表す．そして，このコミュニケーショングラフと先に取得したプロセッサのリソース情報を入力としてグラフ分割アルゴリズムを適用することにより，複数のノードで構成されるグループを作成する．ここでは，スレッド間でメモリバンド幅やキャッシュなどのリソースの競合を起こさないという制約の下で,各プロセッサコアでの処理量を均衡化し，かつグループ間の通信の量をできる限り抑えるようにコミュニケーショングラフの分割を行う．そして，そのグループにもとづいてどのスレッド同士を統合するかを決定する.どのスレッド同士を統合すべきかが決定された後，動的ステージにおいて分割グラフにもとづいてプログラム実行開始時にスレッドを統合する．Thread Tailor では，分割グラフの各グループをカーネルスレッドに対応付け，各ノードをユーザスレッドに対応付けることでスレッド統合を実現する．そして，統合されたスレッドは Thread Tailor によってスケジューリングされることで，プログラムの実行が進む.

3 動的スレッド統合手法

Thread Tailor を用いた実行では，分割グラフにもとづいて，複数のユーザスレッドがカーネルスレッド内にまとめられ，並列化対象関数の実行が開始される．しかしながら，スレッド統合はプログラム開始時，1 度しか行われないため，プログラム実行中にユーザスレッドの組み合わせが変更されることはない．これは，Thread Tailor は並列化対象関数のプロファイリング結果にもとづいてどのユーザスレッドを統合するべきであるかを考慮しているからである．それゆえ，特定の処理区間で，あるユーザスレッドの処理量が変動するようなプログラムを Thread Tailor を用いて実行する場合，各プロセッサコアの負荷の不均衡化および通信量の増大のために実行速度が低下する恐れがある．そこで本研究では，簡易的に動的スレッド再統合を実現するため，バリア同期に着目し，バリア同期時にユーザスレッドを再統合する手法を提案する．提案する手法は

Thread Tailor とは異なり短い区間でユーザスレッドの組み合わせを変更するため，Thread Tailor より高速にプログラムを実行できると考えられる．また,動的にスレッド再統合では既存の Thread Tailor のメモリバンド幅に関する制約は厳しすぎるという問題がある．それは，提案手法は Thread Tailor と異なり，短い区間で取得したプロファイル結果を使用しているため，ある処理区間を実行した際にメモリバンド幅使用量が増加した場合，制約を満たさない可能性が Thread Tailor よりも増加するためである．このことから「グループ内の合計のメモリバンド幅がプロセッサのメモリバンド幅容量を超えない場合に限りノードの交換を許可する」という既存の条件式から「グループ内の最大のメモリバンド幅を持つユーザスレッドがプロセッサのメモリバンド幅容量を超えない場合に限りノードの交換を許可する」という条件式に変更する．

4 実行可能プログラムの拡大

Thread Tailorを用いたプログラム実行では，ユーザスレッドを使用しているためThread Tailorがユーザスレッドをコンテキストスイッチする必要がある．しかしThread Tailorでは，コンテキストスイッチのタイミングをバリア同期時のみに限定しているが，バリア同期時のみではユーザスレッドがループから抜けられず実行が完了しないプログラムが存在する．このようなプログラムには，全てのスレッドがループ内に到達するのを待つ処理，またはあるスレッドの処理が終了するのを他のスレッドがループ内で待つ処理が存在する．そこで，本提案手法では閾値を定め，それを超えるイテレーション回数に到達した場合にのみユーザスレッドをコンテキストスイッチするコードを，各ループ内に挿入することで上記の問題を解決する．これにより，閾値を超える回数ループ内の処理が実行された場合，ビジーウェイトと見なされ，ユーザスレッドがコンテキストスイッチするため，Thread Tailorを用いてより多くのプログラムが実行可能になる．

5 評価

評価対象として，SPLASH-2およびPARSECを使用した．図1には各ベンチマークプログラムと提案手法との組み合わせによる結果がそれぞれ5本のグラフで表されており,通常実行の実行時間を1として正規化している.まず,blackscholesおよびswaptionsは，バリア同期がプログラム実行時に出現しない．そのため,手法(T)および手法(R)で動作に違いが生じず，実行時間に差がなかった.次に，手法(R)により実行速度が向上したbarnes，water-nsquaredではそれぞれ18回および38回のバリア同期が実行され，バリア同期ごとにスレッドが再統合されたことが高速化に繋がった．またfluidanimateは，メモリバンド幅を多量に使用するプログラムであり,手法(T)では分割アルゴリズムのメモリバンド幅の制約が厳しいため,グループ間でノードの交換が一切行われなかった．しかし手法 (R) では，メモリバンド幅の制約を緩和したことにより，適切にグラフが分割されたため,手法(T)に比べ6.0%実行速度が高速化した.最後に,ビジーウェイトが存在し，手法(T)および手法(R)で実行が完了しなかったradiosityおよびcholeskyはプログラムを書き換えることによりThread Tailorで実行が可能となった.

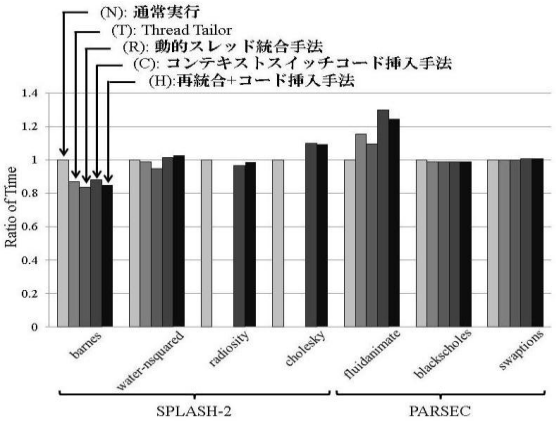

図1: 評価結果

6 おわりに

本研究では，2つの手法を提案した．スレッド再統合手法では，Thread Tailorに比べ最大6.0%の高速化を達成した．またプログラムを改変する手法を利用することにより,デッドロックを防ぎ,既存のThread Tailorでは実行不可能であったradiosity,choleskyの実行が可能になったことを確認した.

6 参考文献

[1] Lee, J. et al.: Thread Tailor: Dynamically Weaving Threads Together for Efficient, Adaptive Parallel Applications, Proc. 37th Annual Int'l Symp. on Computer Architecture, pp. 270-279 (2010).