5H-3

スレッド間データ依存を考慮したパスベーススレッド分割手法の検討

塚本 寛隆 † 大津 金光 † 横田 隆史 † 馬場 敬信 †
† 宇都宮大学工学部情報工学科

1 はじめに

近年普及しているマルチコアプロセッサをより効率的に活用するためには，マルチスレッドコードの生成によるスレッドレベル並列性 (TLP) の抽出が重要となる．

我々は，シングルスレッドコードをマルチスレッドコードに変換する手法として，パスベーススレッド分割手法 [1] を提案している．本手法は，プログラムの実行経路 (パス) の中で最も実行割合が高いパス (#1 パス) に着目し，#1 パスをスレッド分割して投機的なマルチスレッド実行するものであり，プログラムを高速化することを目的としている．本手法ではスレッド間にデータ依存が存在しないようスレッド分割を行うが，スレッド分割数が基本ブロック間のデータ依存の有無に左右されてしまうため，スレッド分割数を大きくできないという問題がある．

そこで本研究では，パスベーススレッド分割手法の改善策として，スレッド間にデータ依存がある場合でも，並列実行を妨げない場合はスレッド分割を可能とする分割手法を検討する．

2 スレッド間のデータ依存を許容するパスベーススレッド分割手法

2.1 従来手法の問題点

マルチスレッド実行においては，スレッド間にデータ依存が存在するとデータの整合性を保つための同期待ちによる性能低下が発生する可能性がある．そこで，従来のパスベーススレッド分割手法では，#1 パス上のコードをスレッド間にデータ依存が存在しないようにスレッド分割を行っている．これはつまり，基本ブロック間に多くのデータ依存が存在する場合，スレッド分割数がスレッドユニット台数に対して少なくなってしまい，大きな性能向上が見込めない可能性がある．

そこでこの問題を改善するために，スレッド間にデータ依存が存在するようなスレッド分割である場合でも，スレッド間に発生する同期待ちにより速度が低下してしまうことがないならばスレッド分割を行うという手法を検討する．またこの改善手法ではスレッドユニットの台数効果が得られるように，マルチスレッド実行における性能向上で重要となるスレッド間の負荷バランスがより均等になるような分割点を選定することも可能となる．

2.2 改善手法のマルチスレッドコード生成手順

マルチスレッドコードの生成の様子を図 1(a) に示す制御フローを持つプログラムと以下の手順で説明する．

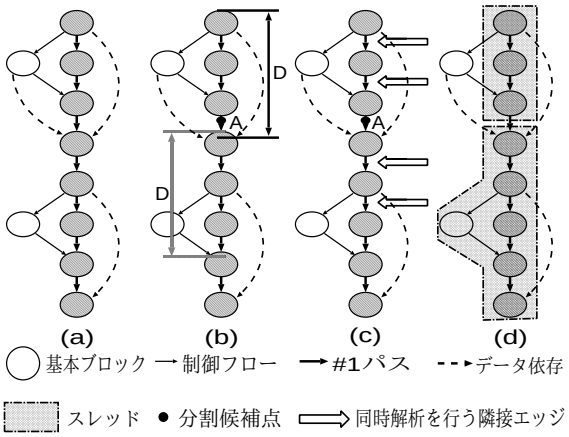

図 1: データ依存を許容するパスベーススレッド分割手法の例

1. パスプロファイルによる#1 パスの決定
2. データフロー解析
3. スレッド分割候補点の列挙
4. スレッドに含める基本ブロックの選定
5. スレッド制御命令の挿入

まず始めに，マルチスレッド化する対象パスを決定するためにパスの実行割合の情報や実行サイクル数をプロファイリングにより取得する．

次に，対象コードのデータフロー解析を行い，対象コード全体のデータ依存を解析する．ここでパス内に関数呼び出しを含む場合は，これをインライン展開することで解析を行う．そして，基本ブロックを単位とした単方向性の有向制御フローグラフから，#1 パスとデータ依存の情報を基にエッジ箇所で複数のスレッドに分割を行う．

本手法のスレッド分割では，スレッド間の負荷バランスを考慮し，対象コードのプロファイリングで得た実行サイクル数とユニット台数から 1 スレッド毎の最適な実行サイクル数を算出しこれを仮に D とする．その上で，図 1(b) のように対象コードの開始点から算出した最適サイクル数 D に直近の基本ブロック間のエッジ A に対して，そのエッジを分割点とした場合に基本ブロック間にデータ依存がない場合はこれを分割候補点とする．この時，基本ブロック間にデータ依存が存在する場合は解析を行い，同期待ちによる速度低下が発生しないならばこれを分割候補点とする．データ依存による同期待ちの解析は，図 2 のようにデータ依存となる値の定義命令と使用命令それぞれについてプログラム実行開始からのサイクル数を見積り，このサイクル数の見積りに基づいてエッジ A で分割した場合のマルチスレッドコード全体の実行サイクル数を見積もる．この実行サイクル数がシングルスレッドコードの実行サイクル数に対して性能向上するようであれば分

Preliminary discussions on path-based thread decomposition method that tolerates inter-thread dependency
†Hirotaka Tsukamoto, Kanemitsu Ootsu, Takashi Yokota and Takanobu Baba
Department of Information Science, Faculty of Engineering, Utsunomiya University (†)

割候補点とする．次の分割候補点を選定する際は，図1(b) の灰色線のように先に選定した分割候補点を開始点として解析を行い，スレッド数がスレッドユニット台数に達するまでスレッド分割を行う．

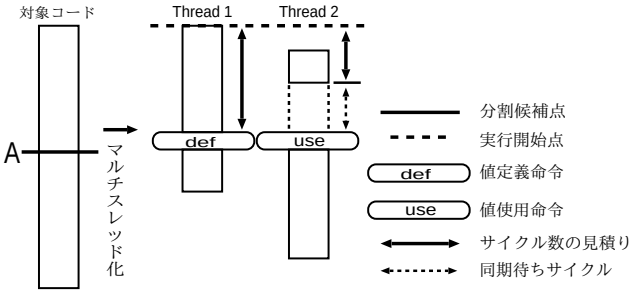

図 2: データ依存による同期待ちの解析の例

ここで本手法では，エッジ毎に基本ブロック間のデータ依存による同期待ちの影響が大きく異なるため，これまで述べた手法により得た分割候補点の前後のエッジに分割候補点を移したものがより速度向上が見込めたり，スレッドユニット台数以下のスレッド分割数でも速度向上が見込める可能性がある．そこで，いくつかのスレッド分割候補点の組み合わせを列挙し，その中から最も性能向上が見込めるものを最終的な分割点とする．具体的には，エッジに対する同期待ちの影響の解析を行う際，図1(c) のように隣接する上下2つのエッジに対しても同様な解析を同時に行う．そして，速度低下するものを除く全エッジに対しても同様の次スレッド分割候補点解析を行う．また，あらかじめ定めたスレッドユニット台数以下の場合に対してもスレッド分割候補点の選定を行うことで，分割候補点の組み合わせを列挙し，その中で最も速度向上が見込めるものを最終的な分割点とする．

これにより#1パスのスレッド分割点が決定したが，マルチスレッド実行においては，あるスレッドで#1パス以外のパスが実行された場合に，投機的に実行されている後続のスレッドを破棄しなければならず，性能低下の原因となる．そこで，#1パス以外のパス内の基本ブロックをスレッドに含めた場合にスレッド間でデータ依存が発生しない場合に限り，投機ミス率を抑えて性能向上を計るためにこれをスレッド内に含める．例えば図1の関数において最も性能向上が見込める分割点が点Aのみ選定された場合，データ依存によりスレッド分割は図1(d) のようになる．

最後に，スレッド制御命令を適当な位置に挿入することで，スレッド間の負荷バランスを均等とし，スレッド間のデータ依存を許容したマルチスレッドコードが生成される．

3 性能評価

評価対象にはSPEC CINT2000の関数を用いる．このスレッド分割対象関数の選定条件として，正しい実行を行うために再帰呼び出しやI/O処理を含まない関数とする．

評価は，シングルスレッド実行サイクル数を改善手法により変換された対象コードのマルチスレッド実行サイクル数で割ることにより求められる速度向上率を用いる．今回の変換では，スレッドユニット台数を4台としている．

評価対象となる関数と速度向上率を以下の表1に示す．入力データセットとしてはtestを用いた．

表 1: 速度向上率

	生成スレッド数	速度向上率
init_chan	2	1.69 倍
start_init	2	1.38 倍
prep_feed_count	2	1.04 倍

- 関数 init_chan

init_chan の#1 パスの実行割合は 100%であった．この関数は，従来のパスベーススレッド分割手法ではスレッド間にデータ依存を含めない分割候補点を選定することができないため，スレッド分割を行えない関数である．init_chan への改善手法の適用では，スレッド分割数を 2 として解析を行った場合にのみ分割点を選定することができた．これは関数内のループにかかるサイクルコストが高いことが原因となり，列挙された分割候補点の組み合わせの中で最も速度向上が見込める組み合わせのスレッド数が 2 となるためである．速度向上率は，スレッド間の負荷バランスが良く，同期待ちが発生しないスレッド分割ができたため 1.69 倍となった．

- 関数 start_init

start_init の#1 パスの実行割合は 62%であった．この関数も init_chan と同様に従来手法では分割することができない関数である．start_init への改善手法の適用では，スレッド分割数を 2 として解析を行った場合のみ分割点が選定された．これは start_init の#1 パスの実行サイクル数が少ないため，スレッドにおける制御命令の割合が高くなってしまうことに起因している．他にも同期待ちが発生しているため，1.38 倍の速度向上となった．

- 関数 prep_feed_count

prep_feed_count の#1 パスの実行割合は 100%であった．この関数を従来の手法で分割するとスレッド分割数は 2 となる．ここで改善手法を適用してみたところ，分割数は同じく 2 となり，分割点も従来手法と同じ位置となった．また，プログラム内に存在するループによりスレッド間の負荷バランスが偏ったため，速度向上が得られなかった．

4 おわりに

本稿では，スレッド間のデータ依存を許容する改善パスベーススレッド分割手法を検討し，これを SPEC CINT2000 に適用した．改善手法では，従来手法では並列化を行うことができなかったコードに対して並列化を行うことができ，速度向上することを確認した．

今後の課題としては，分割点の組み合わせの列挙にかかるコストが大きくなると予想される大規模なプログラムへの手法適用を実現するための計算量の削減の検討が挙げられる．

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 (B)20500047，(C)21500049，(B)21500050）および宇都宮若手萌芽的研究プロジェクトの援助による．

参考文献

[1] 小林崇彦, 大津金光, 横田隆史, 馬場敬信, “パスの実行頻度を考慮したマルチスレッドコード生成手法”, 電子情報通信学会技術研究報告, Vol.106, No.199, pp.7–12, 2006