2J-7

命令を単位と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 † 大川 猛 † 横田 隆史 † 馬場 敬信 †
† 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] を開発している．本手法は，プログラムの実行経路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をする．本手法のスレッド分割は基本ブロック境界に限定して行うが，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点を選定するので，スレッド分割機会が減少して，スレッドユニット台数を有効に活用できない．

そこで本研究では改善策として，命令を単位としたパスベーススレッド分割手法を検討する．従来は基本ブロックを単位とするのに対して，改善手法では命令を単位とすることでスレッド分割機会の増加を図る．本稿では，提案手法の具体的なマルチスレッドコード生成手法について述べる．

2 改善手法の原理

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．しかし，基本ブロック間に多くのデータ依存が存在する場合，スレッド分割数が少なくなり，性能向上が見込めない可能性がある．

そこで，従来手法でのスレッド分割点の選定における基本ブロック境界に限定という制限を無くし，スレッド分割点を命令を単位として選定することで解決を図る．これにより，スレッド分割点となりうる点が大幅に増えて，スレッドレベル並列性を向上させ，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をとりやすくする．

命令を単位としてスレッド分割するために，スレッドユニット台数に基づいた分割数で分割点を決定し，そこにデータ依存が存在する場合は，解消法を適用する．さらにスレッドコードサイズのバランスを調整した後，命令スケジューリングを適用してさらなる効率化を図る．

3 改善手法の手順

改善手法は以下の 6 つの手順から成る．

1. プロファイル情報の取得
2. データフロー解析
3. スレッドユニット台数に基づいた分割点の決定
4. データ依存の解消・許容するためのコード変更
5. スレッドコードのサイズの均等化
6. スレッドコードの命令スケジューリング

3.1 プロファイル情報の取得

本手法では，まずプロファイル情報の取得を行う．対象コードにプロファイル情報の取得用のコードを付加し，1 度だけ逐次実行することで，マルチスレッド化の対象となるコードの#1 パス情報や実行サイクル数などを取得する．

3.2 データフロー解析

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．

3.3 スレッドユニット台数に基づいた分割点の決定

次に，スレッドユニット台数に基づいたスレッド分割数を決定する．これは単純にマルチスレッド化の対象コードの実行命令数をスレッドユニット台数で除算したものである．ここで，各スレッドユニットは投機実行中のメモリへのストアを一時的に書き込んでおくバッファ領域を持っている．そのため，実行命令数の多いプログラムを対象にスレッド分割する場合は，あるスレッドのストアによりバッファが溢れてしまい，性能を著しく低下させてしまう可能性がある．このような対象コードの分割数を決定する場合は，1 つのスレッドに含まれているストア量を減らすために，スレッド分割数を増やすことで一つのスレッドの命令数を減らして，相対的にストア量も減らす．

3.4 データ依存の解消・許容するためのコード変更

スレッド分割点の位置の選定の次に，その分割点候補で分割する場合にスレッド間でデータ依存が存在するかを解析する．データ依存が存在しない場合は，そこがスレッド分割点と決定される．データ依存が存在する場合は，実行時の速度性能の低下を招かないようにデータ依存を解消しなければならない．スレッド分割点上のデータ依存の解消法には，以下の 5 つの方法を用いる．

- レジスタリネーミング
  データ依存関係にあるレジスタを書き換えることで依存を解消する．これは書き換えにより実行に影響を与えない範囲で行う．
- スレッド境界を越えた命令移動による解消
  データ依存関係にある 2 つの命令のうち，どちらか一方をもう一方のスレッドへ移動することで依存を解消する．

Consideration on instruction level path-based thread decomposition method

†Hirotaka Tsukamoto, Kanemitsu Ootsu, Takeshi Ohkawa, Takashi Yokota and Takanobu Baba
Department of Information Systems Science, Graduate School of Engineering, Utsunomiya University (†)

- 値定義命令の fork 前移動・スレッド毎にコピー

  データ依存関係にある 2 つの命令のうち，値定義命令をマルチスレッド実行外のスレッド起動前の位置に移動する．もしくは，値定義命令を依存関係にあるスレッド内にコピーすることで依存を解消する．

- データ依存の許容

  スレッド間のデータ依存が，マルチスレッド実行での速度性能の低下を招かない場合に，スレッド間のデータ依存を許容 [2] する．これは，データ依存が移動やコピーを適用しても解消できない場合に用いる．

これら 5 つの解消法を用いてデータ依存を解消するが，どの解消法を用いても性能低下が起きてしまう場合は，スレッド分割点の位置を移動する．分割点の移動は，性能低下の原因となるデータ依存の領域外に移動する．このとき，移動先にあるデータ依存も同様に解消法を適用する．

3.5 スレッドサイズの均等化

データ依存の解消において，データ依存関係にある命令をスレッド境界を越えて移動する場合や，データ依存領域外に分割点を移動する場合は，スレッド毎の負荷バランスが崩れてしまう可能性がある．そこでスレッド分割点の位置が全て決定した後，各スレッドの負荷バランスをとるために，正しい実行が保証される範囲内で速度性能に影響を及ぼさない命令をスレッド境界を越えて移動する．これにより，より効率的なマルチスレッド実行が可能となる．

3.6 スレッドコードの命令スケジューリング

本手法ではさらなる性能向上を目的として，スレッド分割後に各スレッドコードの命令スケジューリングを行う．命令スケジューリングにおける配置の優先順は以下のようになる．

1. 許容関係にある値定義命令
2. 許容関係にない値定義命令
3. クリティカルパス上の命令
4. その他の命令
5. 許容関係にない値使用命令
6. 許容関係にある値使用命令

4 対象コードにループを含む場合

対象コードにループが含まれる場合は，ループを含めたプログラム全体をスレッド分割する．このとき，パスは上記の命令を単位としたパスベーススレッド分割手法で分割し，ループ部分はイテレーションを単位としてスレッド分割を行う．そこで，パス部分とループ部分の両方をスレッド内に含んでしまうようなスレッド内のループ部分に対して，図 1 のようにループピーリング [3] を適用してパスとして処理する．図 1 では，ループの開始部分の 3 イテレーション分をピーリングによりパスとして処理している．ループ部分のスレッド分割は，ループアンローリングを適用して，イテレーション境界を単位として行う．

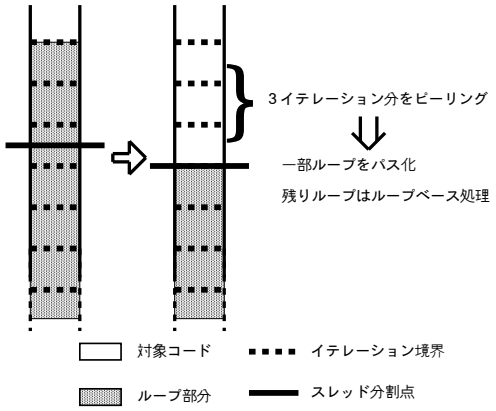

図 1: ループピーリングの例

対象コードにループを含む場合は，プロファイル情報を取得する際に，#1 パス情報以外のループに関する以下の 2 つの情報も取得する必要がある．

- ループのイテレーション回数
- イテレーション毎の実行パス情報

イテレーション回数は，スレッドユニット台数に基づいた分割数を決定する際に，より正確にパスとループのそれぞれの分割数の配分を決定するのに用いる．イテレーション毎の実行パス情報は，ループピーリングを適用する場合に，ピーリングされた数イテレーションがパスベースとして処理されるので，その#1 パス情報として取得する．

5 おわりに

本稿では，マルチスレッドコード生成におけるスレッド分割を命令を単位として行う改善手法について述べた．今後の課題としては，実際のプログラムに改善手法を適用することである．そして，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，速度向上するということを立証する．

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)21500050，同 (C)21500049）の援助による．

参考文献

[1] 小川大仁, 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スレッド分割手法と初期評価”, 情報処理学会第 69 回全国大会講演論文集, pp.1-63–1-64, 2006

[2] 塚本寛隆, 大津金光, 横田隆史, 馬場敬信, “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”, 情報処理学会第 73 回全国大会講演論文集, pp.1-119–1-120, 2011

[3] 中田育男, “コンパイラの構成と最適化”, 朝倉書店, 2009-11