ループ分割により関数境界を越えたパスベーススレッド分割手法の検討

伊里 拓也† 小川 大仁† 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

これまで，我々はループを対象としたマルチスレッド化手法の研究を行ってきた．ループをイテレーション単位でスレッドに分割するこの手法は，イテレーション間の依存がないループであれば高い並列性を得られ大きな速度向上を達成することができたが，依存がある場合などでは並列性が低くなり，速度向上が困難になるなどの問題点があった．また，速度向上を達成できるループはあまり数が多く無く，このマルチスレッド化手法の適用対象は限られていた．

そこでマルチスレッド化する範囲を広げるため，ループ以外の実行パスを対象とするマルチスレッド化手法が考案された [1]．

本研究では，実行パスを対象とするマルチスレッド化手法をどの様な場所に適用すれば、効果的であるかを検討する．

2 実行パスを対象とするマルチスレッド化手法

本章では実行パスを対象とするマルチスレッド化手法の適用手順と手法の問題点について説明する．

2.1 適用手順

本手法は，まず対象となる関数のパスの実行頻度の情報をプロファイリングにより取得する．この時，実行パス上にループが含まれている場合，ループを 1 つのブロックとして扱う．

次に図 1 の左図のように実行頻度が最も高いパス (以下 No.1 パスと呼ぶ) 上の基本ブロック間での依存関係を調べ，命令の移動などにより依存が解消できない所以外で，スレッド間で依存にならないところを分割点とする．分割点が複数ある場合はスレッドサイズが最も均等になる点を分割点とする．

そして，図 1 の右図のように No.1 パスに含まれない基本ブロックの依存を調べ，スレッド間で依存にならないようであればスレッドに含める．図 1 の場合ではブロック 2 がブロック 5 と依存になっているため、ブロック 2 を前スレッドに含めると後続のスレッドと依存関係になってしまう．そのため，ブロック 2 はスレッドに含めず，逆にブロック 8 はスレッドに含めても前スレッドとは依存関係にならないため，ブロック 8 はスレッドに含めることができる．

実行中にブロック 2 の様にスレッドに含めることができなかったブロックを通るパスが実行された場合，後続スレッドを破棄して逐次実行に切り替わるため，できるだけ多くのブロックをスレッドに含めることが望ましい．

2.2 問題点

本手法はこのようにしてマルチスレッド化を行うのだが，いくつか問題点もある．

まず，マルチスレッド化対象となる関数が再帰的になっている場合や，実行パス上にシステムコール命令が存在する場合は，手法が適用できないという問題である．

次に，マルチスレッド化対象となる関数に依存関係により分割点が存在しない場合である．この場合，命令移動などにより最も並列実行される命令数が多い場所を分割点として，依存関係のあるスレッドに分割することで速度向上させることが可能ではあるが，具体的な手法については検討する必要がある．

また，No.1 パスの実行割合が少なく，プログラムの構造上多くのブロックをスレッドに含めることができない場合には速度向上が困難になるという問題点がある．

依存による分割点の偏りやスレッドにループが含まれる場合などに，スレッドサイズが均等にできなくなるという問題もある．スレッドサイズに極端な偏りがあると並列性が低くなり，速度向上の妨げとなってしまう．

他には，対象とする関数のサイズが小さい場合などに，1 スレッド辺りのサイズが小さくなることがある．スレッド制御のオーバーヘッドの問題から 1 スレッド辺り最低 20 命令以上含むことが望ましいが，これを下回ってしまう場合は高速化ができず，逆に速度低下してしまうことがある．このため対象関数はこのままでは高速化ができない．

2.3 関数境界を越えたスレッド分割

スレッドサイズのバランスやスレッドに含めることができる命令数の問題により高速化が困難な場合，対象関数の呼び出し元である関数も含めてマルチスレッド化を行うことで高速化が達成できる場合がある．具体的には，対象関数を呼び出し元に対してインライン展開し，呼び出し元である関数に本手法を適用するというものである．この方法を取ることで，スレッドサイズのバランスを均等にできる分割点が見付かる可能性が増え，また命令数の少ないスレッドは命令数を増やすことができるため，高速化につながると考えられる．

しかしながら問題点もある．本手法ではループは 1 つのブロックと見なして扱うため，関数の呼び出し元がループである場合には関数境界を越えたスレッド分割を行うことができない．

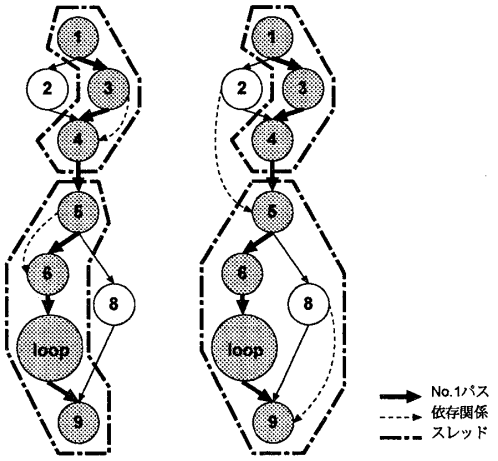

図 1: 実行パスを対象とするマルチスレッド化手法

Preliminary Discussion on a Thread Partitioning Method across Procedures with Loop Division
†Takuya Iri, Hirohito Ogawa, Kanemitsu Ootsu, Takashi Yokota and Takanobu Baba
Department of Information Science, Faculty of Engineering, Utsunomiya University (†)

3 実アプリケーションにおける関数呼び出し

そこで本章では，実アプリケーションを実行した際の関数呼び出しについてのデータを取り，呼び出し元がループである場合がどの程度の割合を占めるか検証する．

3.1 対象アプリケーション

対象とするアプリケーションは *SPEC CINT2000*[2] の 164.gzip，175.vpr，176.gcc，181.mcf，254.gap，256.bzip2，300.twolf と *SPEC CFP2000* の 168.wupwise，171.swim，172.mgrid，173.applu，177.mesa，179.art，183.equake，188.ammp，200.sixtrack，301.apsi の計 17 個である．入力データセットには全て *test* を用いている．

3.2 ループ内関数呼び出し

図 2 は，各調査対象アプリケーションの全関数呼び出しの内，ループ内から呼び出されている回数の割合を算出したものである．最もループ内から呼び出されている回数の割合が多かったのは 164.gzip の 99.934%であり，最も少なかったのが 171.swim の 10.454%であった．アプリケーションによって割合には大きく差があるが，ループ内から呼び出されている割合が 50%以上あるアプリケーションは 9 つあり，調査対象アプリケーションの半数以上であることが分かる．つまりループ内関呼び出しに対応していない本手法は，多くのマルチスレッド化の機会，言わば高速化の機会を失っていることになる．逆に，ループ内関数呼び出しに対応することができれば，より広い範囲をマルチスレッド化することが可能になる．

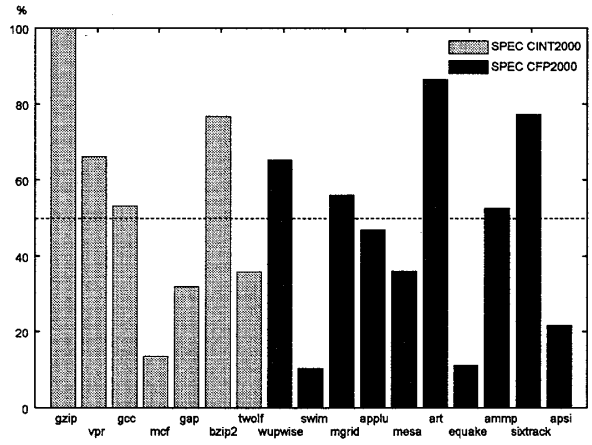

図 2: ループ内関数呼び出し割合

4 呼び出し元ループへの適用

マルチスレッド化による高速化の機会を増やすため，本手法は関数の呼び出し元がループである場合に対処する必要がある．つまり，ループを 1 つのブロックとして扱わず，ループイテレーション内でもパスの実行頻度情報に基づくスレッド分割を行うということである．

図 3 に呼び出し元ループへの適用例を示す．図 3 の左図は，マルチスレッド化対象とする関数とその呼び出し元ループである．対象関数が小さくそのままでは高速化することができないため，図 3 の右図の様に呼び出し元ループに対しインライン展開を行い，呼び出し元ループの基本ブロックを含めてマルチスレッド化する必要がある．

4.1 適用検討

ループへの手法の対応は通常の実行パスへの適用に比べ考慮すべき点が多い．ループは開始地点や終了地点が複数ある場合があり，関数への適用に比べスレッド分割処理が複雑になる．また，イテレーション間での依存がないループは，イテレーション中の全てのスレッドが終了するのを待たず，次のイテレーションでの先頭スレッドを開始することによって効率的な高速化が可能であると考えれ，そのための仕組みを検討する必要がある．

ループイテレーションに対応することによって，イテレーション間に依存があるループや 1 イテレーションが巨大なループなど，イテレーション単位でのマルチスレッド化で高速化が困難なループを高速化できる可能性がある．

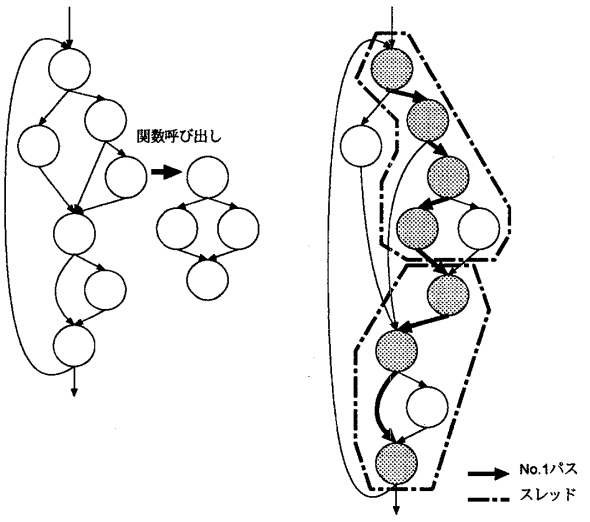

図 3: 呼び出し元ループへの適用例

5 おわりに

本稿では，実行パスを対象とするマルチスレッド化手法の問題点について述べ，関数境界を越えたスレッド分割による解決と，その問題から関数の呼び出し元に関する調査を行った．その結果，関数境界を越えたスレッド分割による効果的なマルチスレッド化を行うためには，実行パスを対象とするマルチスレッド化手法がループイテレーションに対して適用可能な手法にすることが必要であることを示した．

今後は，本手法の具体的な適用手順について検討し，実アプリケーションに対し本手法を適用して評価を行い，イテレーション単位でのスレッド分割手法とどちらがより高速化を達成できるか調査する必要がある．

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 (B)18300014，同 (C)19500037，若手研究 (B)17700047）および宇都宮大学重点推進研究プロジェクトの援助による．

参考文献

[1] 小川大仁，小林崇彦，大津金光，横田隆史，馬場隆信，“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，1K-9，pp.1-63〜1-64，2007

[2] Standard Performance Evaluation Corporation, http://www.spec.org/.