グラフ理論に基づくスレッド分割手法におけるループ展開の適用検討

阿部 武志 † 大津 金光 † 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

近年マルチコアプロセッサが計算機の標準搭載品となり，その性能を最大限に活用するためにシングルスレッドコードをスレッド分割する研究が行われている．

制御フローグラフにおける最小カット集合を求めることでスレッド分割点を決める投機的マルチスレッド化手法 [1](以下，最小カット法) が Troy A. Johnson らによって提案され，この手法によりデータ依存，ロードバランス，スレッド予測のオーバーヘッドを考慮したスレッド分割が可能であった．しかし，ループを分割する場合は 1 イテレーション毎に 1 スレッド生成されるという問題点があり，そのことがスレッドサイズが小さいループを分割しづらくしたり，イテレーション数が多いループの分割時にスレッド制御のオーバーヘッドが性能に大きな影響を与え，スレッドレベル並列度を低めていた．

そこで本研究ではこの問題を解決するために，ループ展開を手法に適用することでループ内でのスレッド分割も可能とし，スレッドレベル並列度を向上させるスレッド分割手法を検討する．

2 最小カット法

最小カット法ではまず始めに，対象となるプログラムの基本ブロックをノード，ブロック間の遷移をエッジとする制御フローグラフを生成する．また，プロファイル情報として 1 ブロックあたりの命令に関する静的な情報，分岐確率，1 関数呼び出しあたりの平均サイクル数，依存関係の情報を得る．

そして，得られたプロファイル情報を基に，制御フローグラフの各エッジに対してそこをスレッド境界とした場合に発生するデータ依存の同期待ちにかかるサイクル数 $\mathcal{D}$ と，投機ミスによって無駄になるサイクル数 $\mathcal{P}$ と，投機ミス率 $freq$ を求め，$\mathcal{W} = \mathcal{D} + \mathcal{P} * freq$ の式によって求まる重み $\mathcal{W}$ を各エッジに割り付けていく．こうして割り付けられた重みに対して最小カットを使うことによって，マルチスレッド実行時のオーバーヘッドが最小となる分割候補点を探し出すことができる．

その後，このままでは負荷不均衡 (*Load Imbalance*) となり性能向上しない可能性があるため，負荷を均等化する処理を施す．その方法は，最小カットで求められる分割候補点とそれ以外の分割候補点とを理論上の実行時間を求め比較する．それ以外の分割候補点は重みの小さいものから順に選んでいき，性能が頭打ちになるまで比較を行うことによって負荷を均等にする．理論上の実行時間とは求めた分割候補点で作られるマルチスレッドコードを実行した場合に，全体でどの程度のサイクル数がかかるかをプロファイル情報や対象スレッドユニット台数から算出した実行サイクル数である．

以上の重みの計算，最小カット，負荷の均等化の作業を繰り返し行うことによってスレッド分割点を求め，マルチスレッドコードを生成する．

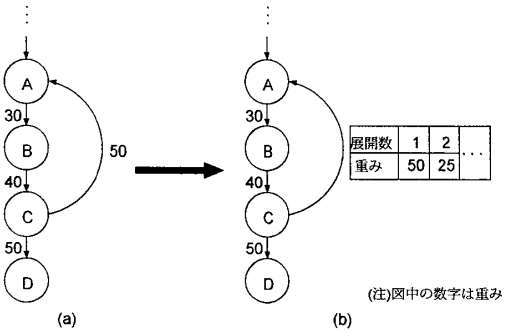

図 1: 展開数毎の重み割り付け

3 ループ展開の適用によるオーバーヘッドの削減

最小カット法にはループ内でスレッド分割点がとられると 1 イテレーション毎にスレッドが生成されてしまうという問題点がある [2]．この問題点によって，1 イテレーションのスレッドサイズが小さいループを分割しづらくしたり，イテレーション数が多いループの分割時にスレッド制御のオーバーヘッドが性能に大きな影響を与え，それによりスレッドレベル並列度を低めていた．

この問題点の解決方法としては最小カット法とループ展開の併用が考えられる．ループ展開を適用することによって，ループを数イテレーション毎に分割することが可能となるので，スレッド制御によるオーバーヘッドが削減し，さらなる速度向上が見込める．

しかし，単純にループ展開を適用すると，コードサイズが増大し，キャッシュミスが増えるのは明らかである．また，制御フローグラフが大きく複雑になるため，最小カット法を適用するのが大変困難なものとなり，現実的な時間内でマルチスレッドコードが生成されない可能性もある．そのため，最小カット法とループ展開の併用を行う際，コードサイズをあまり増大させず，制御フローグラフを複雑にしない方法での問題解決が必要である．

この問題を解決するために考案した，制御フローをほとんど変えずにループ展開の適用を 1 部分に限定して考慮する方法について図 1(a) のような制御フローを持った関数で説明を行う．

図 1(a) の重みは最小カット法を適用したときに割り付けられる重みである．これに対して最小カットを用いて分割点を求めても，ループは 1 イテレーション毎に分割が行われてしまう．

そこで本手法では，図 1 のような制御フローグラフを持つプログラムの場合，ブロック A からブロック B，ブロック B からブロック C に遷移するエッジに対しては従来通り 1 イテレーション毎にスレッドが生成されるときの重みを割り付け，ループの終端ブロック C からループの開始ブロック A に遷移するエッジに限定してループ展開適用の考慮を行う．その方法は，終端ブロックから開始ブロックに遷移するエッジに対して，

Application of Loop Unrolling in Thread Partitioning Method based on Graph Theory
†Takeshi Abe, Kanemitsu Ootsu, Takashi Yokota and Takanobu Baba
Department of Information Systems Science, Graduate School of Engineering, Utsunomiya University (†)

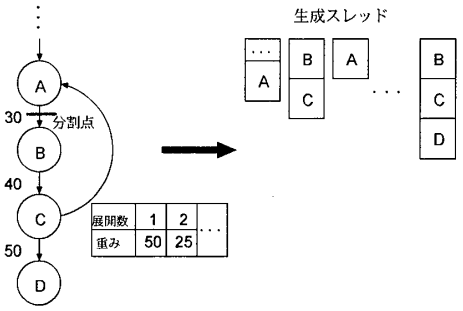

図 2: スレッド分割例 1

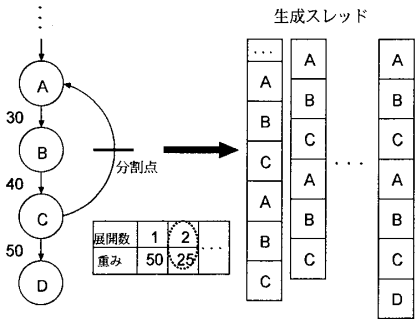

図 3: スレッド分割例 2

そこをスレッド分割点とすることで発生するオーバーヘッド

重みをループの展開数毎に求めて割り付ける．最小カットによって，ブロック A-B，B-C 間でスレッド分割点が取られた場合は従来通り 1 イテレーション毎にスレッドを生成する．

ブロック C-A 間でスレッド分割点が取られた場合は，最小カットの対象となる重みを持つ展開数に応じてループを展開し，その展開数毎にスレッドを生成するようにする．

例えば図 2 左図のように，ブロック A-B 間で分割点が取られた場合，図 2 右図のように，A, BC, A, …, BCD というふうにスレッドを生成し，図 3 左図のように，ブロック C-A 間の展開数 2 で分割点が取られた場合，図 3 左図のように，ABCABC, ABCABC, …, ABCABCD というふうにループを展開数 2 で展開したスレッドを生成するようにする．

この方法によってループ 1 イテレーションの実行サイクル数が小さいときはループ展開が適用され，ループ 1 イテレーションの実行サイクル数が大きいときは 1 イテレーション毎にスレッドが作られたり，それをさらに分割することが可能となる．

4 評価

4.1 評価方法

評価にはスレッドパイプライニングモデルシミュレータである SIMCA[3] を使用する．

評価対象アプリケーションとして *SPEC-CINT2000* の *300.twolf* を用いる．対象とした関数は，*prep_feed_count* である．入力データには *test* を用い，スレッドユニット台数 4 台，8 台，16 台でそれぞれシミュレーションを行った．

バイナリレベルでマルチスレッド化を行うために上記のアプリケーションのコードを SIMCA 用クロスコンパイラ (version 2.7.2.3) に最適化オプション-O3 を適用してコンパイルを行い，それによって生成されたバイナリコードに対して最小カット法，および本改善手法を適用し，マルチスレッド化を行う．

評価は，対象とした関数のシングルスレッド実行サイクル数とマルチスレッド実行サイクル数の比によって求められる速度向上率によって行う．

4.2 評価結果

手法適用上必要なプロファイル情報は，SIMCA 上で対象プログラムのシングルコードを実行し，各基本ブロックのサイクル数，データ依存による同期待ちにかかるサイクル数，分岐確率の計測を行うことによって取得した．その情報をもとに最小カット法，および本手法を適用し，マルチスレッド化を行った．

関数 *prep_feed_count* に最小カット法，および本手法を適用したときの生成スレッド数，速度向上率を表 1 に示す．スレッドユニット台数が 4 台，16 台のときに最小カット法と比較して高速化を達成した．これは最小カット法では 1 イテレーション毎に分割していたループに対してループ展開が適用され，生成スレッド数が減ったことによりスレッド制御のオーバーヘッドが減少し，スレッドレベル並列度が向上したためである．スレッドユニット台数 8 台のときで結果が変わっていないのは，ループ展開を適用すると 1 イテレーション毎に分割した場合に比べ負荷のバランスがくずれてしまい，理論上の実行サイクル数を求めたところ，1 イテレーション毎に分割することが最も速度向上が望める結果となったためである．

表 1: 手法適用による生成スレッド数・速度向上率

	最小カット法		本手法	
	スレッド数	向上率	スレッド数	向上率
4 台	27	2.61 倍	5	2.73 倍
8 台	27	4.12 倍	27	4.12 倍
16 台	27	4.42 倍	14	5.30 倍

5 おわりに

本稿では最小カット法の問題点である，ループの分割は 1 イテレーション毎にスレッドを生成するためスレッドレベル並列度が低い，という問題を解決するためにループ展開の適用を検討した．そして，制御フローグラフを複雑にせずに最小カット法とループ展開を併用する方法として，ループ展開の考慮を 1 部分に限定した方法を考案しその適用検討を行った．評価の結果，本手法適用により，最小カット法のみ適用した場合と比べて，ループ展開によってスレッド制御のオーバーヘッドが削減し，スレッドレベル並列度が向上したため，さらなる速度向上を達成した．

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，同 (C)20500047）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] Troy A. Johnson, et al., “Min-Cut Program Decomposition for Thread-Level Speculation”, Proc. PLDI 2004, pp.59〜70, 2004.

[2] 阿部武志，大津金光，横田隆史，馬場敬信，“グラフ理論に基づくスレッド分割手法の適用検討”，情報処理学会第 70 回全国大会講演論文集，pp.1-99〜1-100，2008

[3] J. Huang, “The SImulator for Multi-threaded Computer Architecture(Release 1.2)”, http://www.cs.umn.edu/Research/Agassiz/Tools/SIMCA/simca.html.