# パスベーススレッド分割手法に基づく自動並列化処理の実装

伊里 拓也† 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

現在主流となっているマルチコアプロセッサの性能を活かすにはマルチスレッドコードによるスレッドレベル並列性（TLP）の利用が重要となる．TLPを効率良く活用するためには自動マルチスレッド化コンパイラが重要となる．

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**を提案している [1]．本手法は，プログラムの実行経路（パス）の内，最も実行割合が高いパス（#1パス）に限定して投機的なマルチスレッド実行を行うことで，制御構造の複雑さや依存変数の削減を行い，TLPをより多く抽出すること．#1パスの実行割合が大きい程，本手法は有利であると言える．本手法はスレッド間にデータ依存を持たせないようにスレッド分割を行うが，データ依存が多く存在するコードではスレッド分割を行うことができない問題があり，この問題の改善が必要となっている．

本研究では，パスベーススレッド分割手法の改善案として，命令移動による依存解消法を用いた改善パスベーススレッド分割手法を提案し自動並列化処理系として実装する．

## 2 依存解消を用いた改善パスベーススレッド分割手法

### 2.1 従来の問題点

パスベーススレッド分割手法は，スレッド間にデータ依存を持たないように#1パス上のコードを複数のスレッドに分割する．言い替えればプログラムを制御フローグラフで表した時，基本ブロック間データ依存が存在しない#1パス上の基本ブロック境界が分割候補点となる．しかしながら，並列化対象となるコードによっては#1パス上においても基本ブロック間データ依存が多く存在し，スレッド分割を行うことができないような場合や分割候補点が偏り性能向上できない場合がある．

本研究ではこの問題を改善するため，スレッド分割の際にスレッド間依存となるような命令を移動させ，スレッド間の依存を解消する処理を用いてパスベーススレッド分割手法の適用可能な範囲を増加させ，TLPの抽出も目指す．

### 2.2 依存解消法

スレッド間データ依存を解消する方法として以下の4つを用いる．

- 変数定義命令のfork命令前移動
- 変数定義命令の後方移動
- 変数定義命令のコピー
- 変数使用命令の前方移動

依存解消処理は，データ依存の変数定義命令と変数使用命令の関係を解析し，プログラムの意味を変えない範囲で基本ブロック境界を越えて命令移動を行うことで，スレッド間データ依存の解消を行う．このため，データフロー解析時に#1パス上で基本ブロック間データ依存となっている命令の移動可能な範囲を解析する．

### 2.3 スレッド分割点の選択

従来通りにスレッド間データ依存とならないような基本ブロック境界をスレッド分割候補点として挙げると共に，分割時にスレッド間依存となるような基本ブロック間依存が存在する#1パス上の基本ブロック境界においても，命令移動範囲の解析からその基本ブロック間依存が分割時に解消可能であればスレッド分割候補点として列挙する．この時，基本ブロック間依存に対して複数の解消法が適用可能ならば，スレッド分割候補点は適用可能な解消法との組み合わせ分列挙する．

スレッド分割候補点の列挙を行った後，そこからマルチスレッド実行時間が最短となり得るスレッド分割点を選択することになる．

ある基本ブロック境界をスレッド分割点として決定した場合，各依存解消処理において命令移動やコピーを行う位置が変化するため，再度スレッド分割候補点を列挙してスレッドを分割していく．

### 2.4 適用手順

命令移動による依存解消を行うパスベーススレッド分割手法の処理手順は，以下のようになる．

1. 対象コードの#1パスの特定
2. データフロー解析
3. スレッド分割点の選択
4. スレッドに含めるパスの選定
5. スレッド制御命令の挿入

まずマルチスレッド化対象コードのパスの実行割合情報をプロファイリングにより取得し，対象コードの#1パスを特定する．この時，#1パス上にループ構造が存在する場合は，実行イテレーション回数も取得する．また，ループを1つのマクロブロックとして扱う．これはループイテレーション中でスレッド分割を行った場合，反復実行中に#1パス以外を通り投機ミスを起こす可能性が高いので，イテレーション中でのスレッド分割は行わないためである．

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．また，#1パス

---

An Implementation of Automatic Parallelization based on Path Based Thread Partitioning Method
†Takuya Iri, Kanemitsu Ootsu, Takashi Yokota and Takanobu Baba
Department of Information Systems Science, Graduate School of Engineering, Utsunomiya University (†)

上に存在する基本ブロック間データ依存となる命令は移動可能な範囲を解析する．

分割した際にスレッド間にデータ依存を持たないような#1 パス上の基本ブロック境界とスレッド間データ依存が解消できる基本ブロック境界をスレッド分割候補点として列挙する．このスレッド分割候補点から，マルチスレッド実行が最速となるようなスレッド分割点を決定し，再度分割候補点の列挙，スレッド分割点の決定と実行時間を縮めることが不可能になるまで分割していく．

スレッド分割点が決定した段階では，各スレッドは#1 パス上の基本ブロックのみを含んでいるが，ここで#1 パス上以外のパスを各スレッドに含める処理を行う．#1 パス以外のパスもスレッドに含めることにより，投機ミスの発生を抑え高速化効果を高めることができる．

最後に，スレッド制御命令を適切な位置に挿入し，スレッド間依存となる命令の移動を行うことでマルチスレッドコードの完成となる．

## 3 自動並列化システム

### 3.1 実装方針

提案した改善手法の自動並列化システムへの実装方針を以下に示す．

- 命令セットに依存しない並列化処理
  並列化対象コードを中間表現にデコードし，この中間表現で並列化を行った後、エンコードして出力することで命令セットに依存しない並列化処理を行う．
- #1 パス上での汎用的な並列化処理
  並列化を行う部分が関数全体、またはその関数の#1 パス上の一部であったとしても、プログラムの構造に依存することなく汎用的に並列化を行える処理を実装する。

### 3.2 自動並列化システム構成

図 1 は実装したシステムの構成図である．このシステムは instruction decoder と path parallelizer に分かれている．また，path parallelizer は data flow analyzer, thread partitioner，code generator を持っている．

まず，並列化対象となるバイナリコードを入力し，instruction decoder で中間表現に変換する．この中間表現データと#1 パス及びループイテレーション回数のデータを data flow analyzer に入力し，データフロー解析とこの解析に基づく命令移動可能範囲の解析を行う．この解析結果データと対象アーキテクチャのリソースサイズのデータを基に thread partitioner ではスレッド分割候補点の列挙及びスレッド分割点及び依存解消法の選択，スレッドに含めるパスの選定処理を行い，各スレッド構成のデータを生成する．このスレッド構成データを基に code generator では，スレッド制御命令の挿入及び命令移動によるスレッド間データ依存の解消を行い，マルチスレッドコードを生成する．

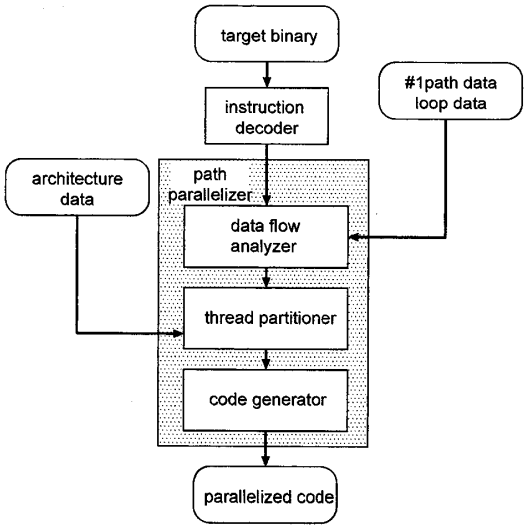


図 1: 自動並列化システム構成図

## 4 評価

実装した自動並列化システムの評価を行うために，表 1 に示す SPEC CINT2000 アプリケーションの関数を対象に並列化を試みた．これらの関数は従来のパスベーススレッド分割手法では，基本ブロック間データ依存により#1 パス上にスレッド分割候補点が見付からず，並列化を行うことができなかったものである．

並列化コードを実行するアーキテクチャのリソースが十分に存在するものとして自動並列化システムにデータを入力したところ，表 1 に示す通りに逐次実行コードに対して速度向上率を得た．

表 1: 適用結果

| 関数名 | 速度向上率 |
|---|---|
| fill_window（164.gzip） | 2.22 |
| _getmode（197.parser） | 1.11 |
| strtol（300.twolf） | 1.16 |

## 5 おわりに

本稿では，改善パスベーススレッド分割手法によるプログラムの自動並列化システムの実装を目指し，依存解消を用いたスレッド分割についての特徴および処理手順について述べた．実装したシステムでは，従来手法では並列化を行うことができなかったコードを並列化することができ速度の向上を確認した．

今後の課題として，より多くのプログラムに適用して有効性の検証を行うことが挙げられる．

**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 (C)20500047, 同 (C)21500049, 同 (C)21500050) 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の検討”, 電子情報通信学会コンピュータシステム研究会 (CPSY), 信学技報, Vol.106, No.199, pp.7–12, 2006