

実行時プロファイルを用いた Haskell 向け自動並列化の実装

青江 光敏[†] 千葉 雄司[‡] 土居 範久[†]

中央大学大学院 理工学研究科 情報工学専攻[†]

中央大学 研究開発機構[‡]

1 はじめに

近年マルチコアプロセッサが普及し始め、並列プログラミングの重要性が増してきている。しかし、並列プログラミングは逐次プログラミングに比べ難しい。プログラムを効果的に並列化するには、スレッド生成のオーバーヘッドを慎重に見極めてプログラムを作成する必要がある。また、非正格評価を行う言語においては並列化によって停止性が変化しないよう気を配る必要がある。このため、手動による並列化は手間が大きく、言語処理系による自動並列化への要求が高まっている。本研究では非正格評価を行う純粋関数型言語 Haskell を対象とした自動並列化技術を提案する。Haskell における既存の自動並列化技術では、正格性解析を行い並列化する方法[2][3]があるが、正格性解析のみで効果的な並列プログラムを構成するのは困難である。そこで本研究では正格性解析に加えて、実行時プロファイルを用いて実際に並列化の効果が大きい部分を求め、求めた部分を並列化する手法を提案する。

2 プログラミング言語 Haskell と並列プログラミング

Haskell は非正格評価や静的な強い型付けを特徴とする純粋関数型言語のひとつである。Haskell 向けの標準的な言語処理系である Glasgow Haskell Compiler (GHC)は P.W. Trinder らが提案した並列プログラミングライブラリおよびランタイムシステム Glasgow Parallel Haskell (GpH) [1]を採用している。

GpH の並列プログラミングでは、プログラマが並列化したい部分に並列処理用の関数を挿入することで並列化を行う。GpH は評価戦略 (evaluation strategy)を導入し、複雑な並列プログラムを簡潔に記述可能にする。本研究では評価戦略を用いて次の 4 つの代表的な並列

プログラミングモデルを用いて並列化を行う。

1. 分割統治並列化
2. データ主導並列化
3. 生産者・消費者並列化
4. パイプライン並列化

分割統治並列化とは、問題をいくつかの小さい部分問題へと分割し、各部分問題を並列に計算し、解を集めてもとの解とする手法である。データ主導並列化はリストや配列のようなデータ型において、各要素の計算を並列に行う手法である。また、生産者・消費者並列化は、ある計算がデータを生成し、一方でそのデータを消費する計算があるとき、両者を並列に計算する手法である。最後に、パイプライン並列化は、値に対して複数の関数を順に適用していく場合に、それぞれの関数適用を並列に行う手法である [1]。

3 非正格評価と並列性

Haskell のような非正格評価を行う言語では、効果的な並列化を行うために、部分的に正格評価を行う必要がある。しかし、このとき必要以上に正格評価を行うと、並列化によってプログラムの停止性が変化してしまうことがある。たとえばリストの各要素に関数を適用する関数 `map` は渡されたリストに対して非正格であるが、並列化された `map` である `parMap` は各要素が弱頭部正規形になるよう指定すると、リストに対して正確になり、停止性が変化する。本研究では、どの程度正格評価が必要か判断するために、GHC の正格性解析を用いる。GHC の正格性解析は Demand Analysis[5]を行う。

具体的には、並列化を検討する関数の各引数についてその正格性を検査し、正格であればその引数を正格評価できるため、単純に並列化できる。そうでなければ正格評価を行うとプログラムの停止性が変化する。他の方法で並列化するか、該当部分の並列化をあきらめ、他の部分の並列化を検討する。

4 並列化箇所の特定手順

自動並列化の手順は図 1 のとおりである。図 1 の並列化プログラムモジュールでは、実行時プロファイルの情報から並列化の効果が見込める

Automatic Parallelization for Haskell using Runtime Profiling
Mitsutoshi AOE[†], Yuji CHIBA[‡], Norihisa DOI[†]

[†]Information and System Engineering Course, Graduate School of Science and Engineering, Chuo University

[‡]Research and Development Initiative, Chuo University

箇所を順序付けして列挙する．優先順位の高いほうから順に，正格性解析を行い 4 つのモデルに合致するかどうか検査する．

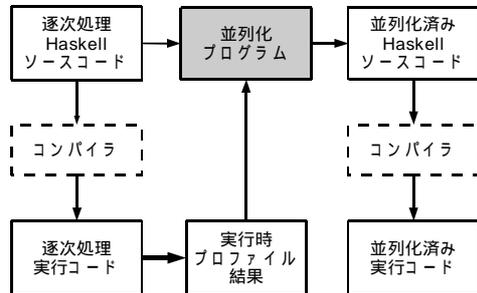


図 1: 並列化処理の流れ

実行時プロファイルからわかる情報は，呼び出し木全体と，木のノードごとに呼び出し回数と消費時間・消費メモリが記録されている．これをもとにコストの大きいノードから順に並列化可能かどうか判定し，可能であれば並列化する．

並列化可能かどうかの判定は次の手順で行う．まず，根のノードから単位呼び出しあたりの消費時間が大きいノードを探し，そのノード内で子ノードの呼び出しを分割できるか検査する．K 子ノードの呼び出し同士に依存関係がある場合は生産者・消費者並列化あるいはパイプライン並列化を適用できるか検討する．依存関係がない場合は分割統治並列化を検討する．また，map や zipWith のようなデータ手動並列化に適した特定の関数を使用している場合は，停止性の変化が起きないか検査して parMap や parZipWith に置き換え可能か検討する．

5 実装

実装したのは図 1 の並列化プログラムモジュールである．このモジュールは 2 つの機能からなる．ひとつはテキストファイルとして書き出されたプロファイル結果を解析するモジュールで，もうひとつは，得られた結果を元に並列化箇所の特定を行うモジュールである．どちらも Haskell を用いて実装した．

6 実験と評価

実験の対象とするプログラムには Haskell 用のベンチマーク集 nofib[4]から有限要素法を用いたトラス構造の解析を行うプログラム(fem)と，セル中の粒子を計算するプログラム(pic)につい

て計測した．コンパイラは GHC の最新 HEAD ブランチ(6.7)を使用し，計測 PC はハイパースレッディングを搭載した 2.8GHz Pentium 4 プロセッサと 1GB のメモリを搭載している．表 1 に実験結果を示す．

表 1: 実験結果

プログラム	逐次処理	並列処理	性能向上
fem	5.46 秒	5.38 秒	1%
pic	9.06 秒	6.33 秒	30%

以上のように，自動並列化の結果，最大で 30% の性能向上が見られた．

7 まとめ

本研究では純粋関数型言語 Haskell を対象とした自動並列化手法を提案した．従来の静的解析のみを用いた自動並列化では，スレッド生成のオーバーヘッドの見積もりが難しく，効果的な並列化が困難であった．提案手法ではこの問題を解決するべく，静的な解析に加え実行時プロファイルの情報から並列化すべき部分を特定した．これにより最大で 30% の性能向上が確認できた．

参考文献

- [1] P. W. Trinder, K. Hammond, H-W Loidl and S.P. Jones, "Algorithm + Strategy = Parallelism", Journal of Functional Programming 8(1), pp. 23-60, Jan 1998.
- [2] G.L. Burn, "Implementing the Transformer Model of Reduction on Parallel Machines", Journal of Functional Programming 1(3), pp 329-366, July 1991.
- [3] S. Mintchev, "Hyperstrictness and the Parallel Evaluation of Lazy Functional Programs", Proceedings 4th Workshop on Parallel and Distributed Processing: WP&DP'93, Bulgarian Academy of Sciences, pp. 260-268, March 1993.
- [4] W. Partain, "The nofib Benchmark Suite of Haskell Programs", In Functional Programming, J Launchbury and PM Sansom, eds., Workshops in Computing, Springer Verlag, pp. 195-202, July 1992.
- [5] M. Schütz, "Analyzing demand in non-strict functional programming languages", Dissertation, Johann Wolfgang Goethe-Universität Frankfurt, 2001.