Vol.2017-ARC-225 No.35
Vol.2017-SLDM-179 No.35
Vol.2017-EMB-44 No.35
2017/3/10

自動並列化コンパイラのコンパイル時間短縮のための実行プロファイル・フィードバックを用いたコード生成手法

藤野 里奈[1,a)]　韓 吉新[1]　島岡 護[1]　見神 広紀[1]　宮島 崇浩[2]　高村 守幸[2]　木村 啓二[1]　笠原 博徳[1]

概要：粗粒度タスク並列，ループ並列，近細粒度並列を階層的に組み合わせるマルチグレイン並列処理による自動並列化は，従来の単体ループ並列化に対しより高い並列性の利用が可能となる．特にこれまで並列化が困難と考えられていたループが存在しない条件分岐と基本ブロック及び関数から構成される自動車エンジン制御のプログラムに対しても粗粒度タスク並列化により性能向上が可能であることが確かめられている．このとき，自動車エンジン制御のようなセンサ入力に伴う条件分岐を多数含む組み込みハードリアルタイム制御システムでは，条件分岐方向を確実に予測することが困難なため，コンパイル時のタスク処理時間の高精度推定が難しい．このため，このような場合には実行時プロファイルを基に性能チューニングを行い，効果的な並列化が実現できるようになってきている．プロファイル・フィードバックによる自動並列化では，プロファイルを用いてタスク融合の最適化を行う都度プロファイリングを行いプロファイル情報をフィードバックする度に，プログラム全域の並列性を抽出し，翻訳を繰り返し行う必要がある．プログラムサイズが数百万行に及ぶ実プログラムを繰り返しコンパイルすると，翻訳時間が数時間から数十時間を要する場合もあり，プログラム翻訳時間の削減がソフトウェア生産性の面から重要である．本稿では，プロファイル情報を用いた再解析・再リストラクチャリングを可能な限り省略し，自動並列化コンパイラの翻訳時間の短縮を行う手法を提案する．更に，マルチグレイン並列処理でプログラムの自動並列化を行う OSCAR マルチグレイン自動並列化コンパイラに，本手法を実装し，評価を行う．翻訳時間の評価では，従来手法では 6935.19[s] の翻訳時間がかかる粗粒度タスク数が多くループのほとんど無いテストプログラムが，本手法適用後，翻訳時間が 1.27[s] まで削減された．また，並列化コードの実行時間の評価では，従来手法と本手法で生成した並列化コードの実行時間の差が 3%以下となり，従来手法と本手法で並列化性能に差異はないことを確認した．

キーワード：自動並列化コンパイラ，プロファイル・フィードバック，翻訳時間，高速化

1. はじめに

マルチコアプロセッサシステムはデスクトップコンピュータから高性能端末，組み込みシステムまで広く使われている [1]．しかしながら，並列プログラミングは難易度が高いだけでなく，使用者の所望する性能を得るためには実行時プロファイルを基にした並列化チューニングを繰り返す必要がある．特に，自動車エンジン制御に代表される組み込みハードリアルタイム制御システムでは，センサ入力に伴う条件分岐を多数含み，タスク処理時間の高精度推定が困難であるため，数度にわたり，実行時プロファイルを参照しながら並列化チューニングが行われている．そのため，並列処理のために工数を多く費やさなければならず，並列ソフトウェアの生産性向上が大きな課題となっている [2]．

並列ソフトウェアの生産性向上のため，筆者等は OSCAR マルチグレイン自動並列化コンパイラを開発している [4–6]．OSCAR コンパイラではプログラム中の粗粒度タスク並列処理，ループレベル並列処理，近細粒度並列処理を組み合わせたマルチグレイン並列処理を実現している．マルチグレイン並列処理により，従来のループ並列処理のみでは並列性抽出できなかった制御系のプログラムからも並列性抽出が可能となる [7]．

しかしながら，マルチグレイン並列処理を行うためには，プログラム全域の並列性を抽出し翻訳する必要があるため，大規模プログラムの翻訳に多大な時間がかかる場合があるという問題がある [8]．

OSCAR コンパイラにおけるプロファイル・フィードバックを用いた高精度並列化コード生成手法は，実行時プ

[1] 早稲田大学 理工学術院 情報理工学科
Dept. Computer Sci. & Eng., Waseda University.
[2] オスカーテクノロジー株式会社
Oscar Technology Corporation.
[a)] rfujino@kasahara.cs.waseda.ac.jp

Vol.2017-ARC-225 No.35
Vol.2017-SLDM-179 No.35
Vol.2017-EMB-44 No.35
2017/3/10

ロファイルを自動取得するパス 1 と，プロファイルを基に並列化チューニングを行い並列化コードを生成するパス 2 から構成されている．マルチグレイン並列化を実現するため，実行時プロファイルを自動取得する際と，プロファイルを基に並列化コードを生成する際に，既存の逐次プログラムの翻訳を行う必要がある．また，プログラムを実行するターゲットマシンの CPU の世代交代時，並列化コード外の関数変更時など，外部要因が原因となりプロファイル結果が変わる場合も，その都度既存の逐次プログラムの再翻訳を行う必要がある．そのため，プログラムサイズが数百万行にも及ぶ実プログラムなどの翻訳に多大な時間を要するプログラムにおいては，時として翻訳時間に数時間から数十時間を要する場合があり，ソフトウェア生産性の面から自動並列化コンパイラの翻訳時間の削減は重要となる．

本稿では実行時プロファイル・フィードバックを用いた自動並列化翻訳時間削減手法を提案する．本提案手法は従来と同様の 2 パス構成であるが，パス 2 において逐次プログラムの再解析・再リストラクチャリングは可能な限り省略する．パス 2 でフィードバックされたプロファイル情報を基にスケジューリングのみ行うことにより，パス 2 の翻訳時間の大幅な削減を可能とする方法を提案する．また，性能評価ではプロファイル情報を基に並列化チューニングを行うことが多い組み込みハードリアルタイム制御システムを模した粗粒度タスク数が多くループのほとんど無いテストプログラムを用い従来と比較して翻訳時間の大幅な削減が可能となることを示す．更に，本手法により並列化性能が従来手法と変わらないことを確認し，CPU の世代交代によるプロファイル情報の更新をシミュレーションするため，ODROID-XU3 [9] 上で周波数制御を行った際，各周波数で実行時間が従来手法と変わらないことを示す．

本稿の構成を以下に示す．まず第 2 節でプロファイル・フィードバックを用いた自動並列化手法について，第 3 節でプロファイル・フィードバックを用いた並列化コンパイル高速化手法について述べる．次に，第 4 節で提案手法を用いたテストプログラムの翻訳・並列化性能の評価について述べ，第 5 節でまとめを述べる．

2. プロファイル・フィードバックを用いた自動並列化手法

本章では，OSCAR コンパイラがプロファイル・フィードバックを用いて逐次オリジナルソースコードから，並列化コードを生成する自動並列化手法について説明する．

2.1 OSCAR コンパイラの概要

OSCAR コンパイラは，C や Fortran77 で記述された逐次プログラムを入力とし，マルチグレイン並列化を行う自動並列化コンパイラである．マルチグレイン並列化は，粗粒度タスク並列処理，ループレベル並列処理，近細粒度並列処理を階層的に組み合わせた並列化手法のことである．マルチグレイン並列処理により，ソースプログラムは基本ブロック (BB)，ループなどの繰り返しブロック (RB)，関数呼び出しのサブルーチンブロック (SB) の 3 種類に分割される．これら 3 種類のブロックはマクロタスク (MT) と呼ばれる．OSCAR コンパイラの粗粒度並列化では，MT 間の依存関係とコントロールフローを解析し，マクロフローグラフ (MFG) を生成する．更に最早可能実行条件に基づいた解析により，MFG から MT 間の並列性を抽出し，タスクスケジューリングを行いつつマクロタスクグラフ (MTG) として並列性を表現する．そして OSCAR コンパイラは条件分岐などの実行順不確定性の高い並列化階層にはダイナミックスケジューリングを適用し，そうでない階層にはスタティックスケジューリングを適用する [4–6]．

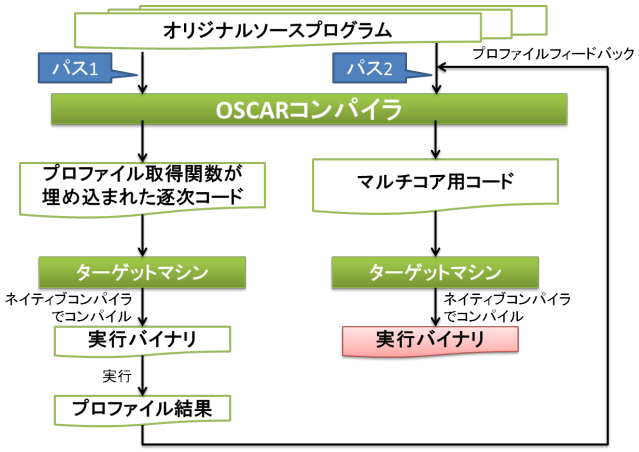

図 1 OSCAR コンパイラにおけるプロファイル・フィードバック機能のコンパイルフロー

2.2 プロファイル・フィードバックを利用した並列化コード生成フロー

OSCAR コンパイラのプロファイル・フィードバックを用いて高精度並列化コード生成を行う手法は，実行時プロファイルを自動取得するパス 1 と，プロファイルを基に並列化チューニングを行い，並列化コードを生成するパス 2 から構成されている [10]．その処理フローの図を図 1 に示し，また以下の節でそれぞれのパスについて詳しく説明する．

2.2.1 従来手法のパス 1

パス 1 では，逐次プログラムであるソースプログラムを OSCAR コンパイラに入力し，OSCAR コンパイラはプロファイル自動取得のための関数を埋め込んだ逐次コードを出力する．このとき，プログラムの解析・リストラクチャリングは行うが，プロファイル取得のための逐次コードが生成される．生成された逐次コードをターゲットマシン用にコンパイルし，実行バイナリを得，ターゲットマシン上で実行する．プロファイル結果としては，主に各 MT のコ

Vol.2017-ARC-225 No.35
Vol.2017-SLDM-179 No.35
Vol.2017-EMB-44 No.35
2017/3/10

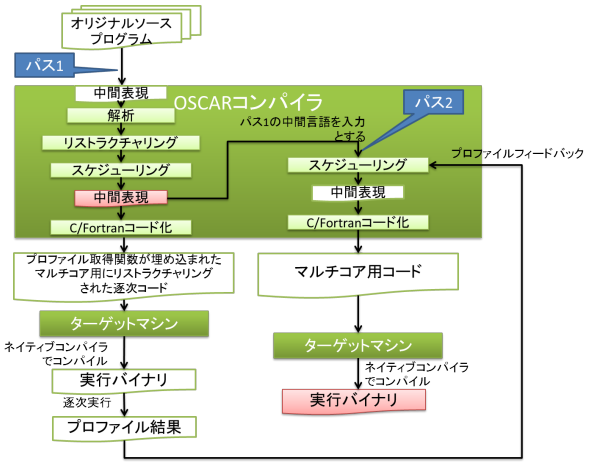

図 2　提案手法のプロファイル・フィードバック機能のコンパイルフロー

ストと何回当該 MT を実行するかの情報が得られ，これをパス 2 で使用する．

2.2.2　従来手法のパス 2

パス 2 では，逐次プログラムであるソースプログラムに加え，パス 1 で得たプロファイル情報を OSCAR コンパイラへ入力する．OSCAR コンパイラはプログラムの解析・リストラクチャリングを再度行い，パス 1 で得たプロファイル情報をフィードバックしスケジューリングに反映させ，指定したコア数用のマルチコア用並列化コードを出力する．その後，ターゲットマシン上で並列化コードをコンパイルし，最終的に得たい実行バイナリを得る．

3.　プロファイル・フィードバックを用いた並列化コンパイル高速化手法

第 2 節で述べた従来手法では，コストを再計測するたびに OSCAR コンパイラによるソフトウェア全域の解析及びリストラクチャリングを適用しマルチグレイン並列化を行う．実行ターゲットマシンの CPU の世代交代時や，並列化コード外の関数変更時など，並列化コードの MT 間の依存やコントロールフローに変更がない場合も，現状ではソフトウェア全域の解析及びリストラクチャリングを再度行う．しかしながら，このような再解析及びリストラクチャリングにより翻訳に多大な時間がかかる場合，ソフトウェア生産性の低下の原因となる．そのため，本章では並列化コードの MT 間の依存やコントロールフローに変更がなく，コストのみが変化した場合，スケジューリングのみをパス 2 で行い，再翻訳し，並列化コードを生成する手法を提案する．図 2 に本手法のフローを示し，また以下の節でそれぞれのパス及び制限事項について詳しく説明する．

3.1　提案手法のパス 1

提案手法のパス 1 は従来手法とほぼ同様であるが，OSCAR コンパイラの出力が異なる．OSCAR コンパイラに逐次ソースプログラムを入力し，指定したコア数に適した解析，リストラクチャリング，及びスケジューリングを行った後，プロファイル取得関数を埋め込んだ逐次コードを生成する．ただしパス 1 が出力する中間表現は，パス 2 で再解析・再リストラクチャリングを行わないことを目的とするため，並列化階層情報 [11] やマクロタスクグラフ情報が埋め込まれている．OSCAR コンパイラが出力した逐次コードをターゲットマシン上のネイティブコンパイラでコンパイルし，実行バイナリを得る．そして，逐次実行し，プロファイル結果を自動取得する．

3.2　提案手法のパス 2

提案手法のパス 2 は従来手法と入力が異なる．パス 1 で解析・リストラクチャリング・スケジューリングを終え，OSCAR コンパイラが生成した中間表現と，プロファイル結果を入力とする．パス 2 の入力となる中間表現はパス 1 で指定したコア数に最適化されたリストラクチャリングされた構造になっているため，再解析や再リストラクチャリングは行わない．プロファイル・フィードバックによって得られたプロファイル情報を基にスケジューリングのみを行い，パス 1 で指定したコア数用のコードを生成する．その後，ターゲットマシン上でネイティブコンパイラを用いてコンパイルし，目的の実行バイナリを得る．

3.3　制限事項

本手法は従来手法と比較すると，以下の制限となる事項が存在する．

(1) パス 2 では使用コア数を変更できない
(2) パス 2 では粗粒度並列処理を適用する階層の構造を変更できない

これらはパス 2 では粗粒度タスクスケジューリング以外の処理を行うことができないため，パス 1 で適用するリストラクチャリング手法及び解析で決定する並列化階層を変更できないことに起因する．

本制限事項のため，本手法は並列化対象プログラム自身に変更を加えず，プロファイルを繰り返し取得し，並列化コードを生成するときに使用することを対象としている．

本手法における性能上の注意点として，この制限事項により，従来手法と比べて，並列性を十分に抽出できない可能性が考えられる．

4.　提案手法を用いたテストプログラムの翻訳・並列化性能の評価

本節で提案手法の性能評価結果について述べる．評価対象はテストプログラムに対する OSCAR コンパイラにおけるパス 2 の翻訳時間及びターゲットマシン上での実行時間である．

Vol.2017-ARC-225 No.35
Vol.2017-SLDM-179 No.35
Vol.2017-EMB-44 No.35
2017/3/10

表 1　OSCAR コンパイラ翻訳時間取得マシンの仕様

OS	Ubuntu 14.04 LTS
CPU	Intel Xeon E5-2667 v4 (3.2GHz)
コア数	16
L1 D-Cache	32K
L1 I-Cache	32K
L2 Cache	256K
L3 Cache	25,600K

表 2　ODROID-XU3 の仕様

OS	Ubuntu MATE 15.04 LTS
CPU	ARM Cortex-A15(2.4GHz)
	ARM Cortex-A7(1.4GHz)
コア数	16
L1 Cache	32K
L2 Cache	512K + 2MB

表 3　評価プログラムの情報

	テストプログラム A	テストプログラム B
プログラム行数	1,720,919	334,297
関数数	95,867	384
ブロック数	9,350	6,270
MT 数	69	63

4.1　評価環境

本評価では，OSCAR コンパイラの翻訳時間の評価を Intel の 16 コア CPU である Xeon E5-2667 v4 を 1 基搭載したマシン上で行った．また，並列化したプログラムの実行時間の評価を ARM Cortex-A15 と ARM Cortex-A7 を集積した ODROID-XU3 [9] 上で行った．評価の際は，動作周波数を 200MHz，400MHz，600MHz，800MHz，1GHz，1.2GHz，1.4GHz にそれぞれ設定した．ODROID-XU3 が 4 コアであるため，OSCAR コンパイラで 2 コア用と 4 コア用の並列化コードの生成を行い，ODROID-XU3 上で 2 コアと 4 コア実行時の評価を行った．Intel Xeon E5-2667 v4 を搭載したマシン及び ODROID-XU3 の仕様をそれぞれ**表 1**，**表 2** に示す．

4.2　評価プログラム

評価プログラムとして，制御プログラムのように粗粒度タスク数が多いがループがほとんど無く，翻訳に長時間かかる大規模プログラムである「テストプログラム A」，「テストプログラム B」を使用した．各プログラムの特徴であるプログラム行数，関数数，ブロック数，スケジューリング対象 MT 数を**表 3** に示す．

4.3　並列化翻訳時間の評価

この節ではパス 2 において，従来手法と本手法を使用したときの OSCAR コンパイラの翻訳時間について述べる．テストプログラム A における翻訳時間評価結果を**図 3**，テストプログラム B における翻訳時間評価結果を**図 4** にそれぞれ示す．

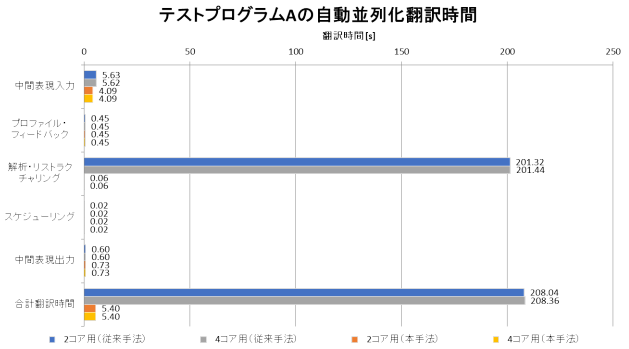

図 3　テストプログラム A の翻訳時間評価

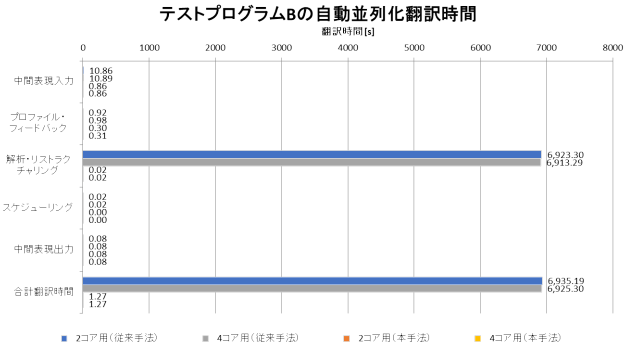

図 4　テストプログラム B の翻訳時間評価

OSCAR コンパイラでの翻訳は，図 2 に示した通り，中間表現の入力部，解析・リストラクチャリング部，スケジューリング部，プロファイル・フィードバック部，中間表現出力部に分けて考えることができる．そのため，本評価では以上の分類に分け，翻訳時間の実測を行った．また，2 コア用，4 コア用の並列化コード生成を行い，指定コア数の違いにより翻訳時間に差異が出るかどうかの確認も行った．

本手法の性質上，本手法のパス 2 の解析・リストラクチャリング部の翻訳時間が非常に短縮されることが予想される．図 3 の解析・リストラクチャリングの項で示した通り，テストプログラム A で従来手法を用いると 2 コア用で 201.32[s]，4 コア用で 201.44[s] だったものが，本手法では 2 コア用 4 コア用共に 0.06[s] まで削減された．またテストプログラム B では，図 4 の解析・リストラクチャリングの項で示した通り，従来の手法では 2 コア用で 6923.30[s]，4 コア用で 6913.29[s] だったものが，本手法により 2 コア用 4 コア用共に 0.02[s] に削減された．テストプログラム A・B では 2 コア用 4 コア用関わらず，解析・リストラクチャリング部の時間が大幅に削減していることが確認できた．

また，図 3，図 4 のスケジューリングの項の通り，従来手法でもスケジューリング部は 2 コア用 4 コア用関わらずテストプログラム A で 0.02[s]，テストプログラム B で 0.02[s] と小さく，スケジューリング部が大規模プログラム

Vol.2017-ARC-225 No.35
Vol.2017-SLDM-179 No.35
Vol.2017-EMB-44 No.35
2017/3/10

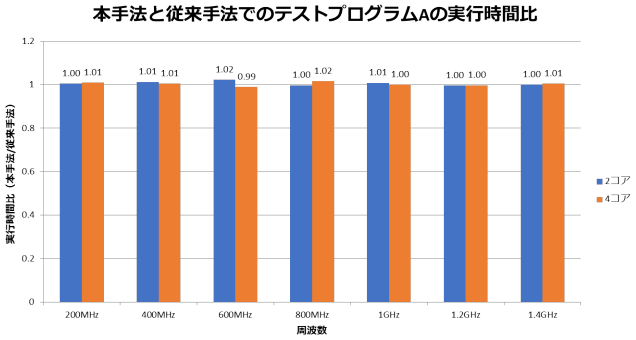

図 5　テストプログラム A の各周波数における実行時間評価

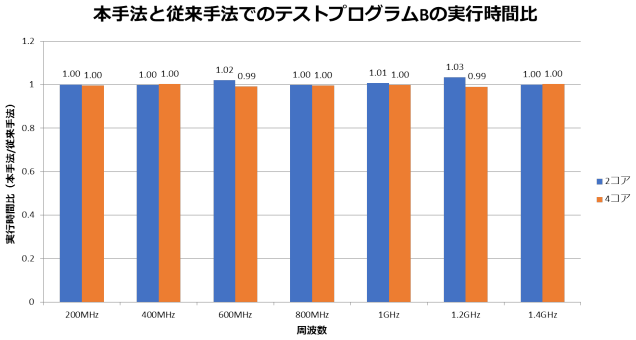

図 6　テストプログラム B の各周波数における実行時間評価

の合計翻訳時間に与える影響は無視できることがわかる．本手法のパス 2 で計上されている合計翻訳時間は，入出力である中間表現入力・出力，プロファイルフィードバックであるといえる．

プログラムによってパス 2 の実行に従来は数時間かかっていたものが，数秒単位になるため，プログラム生産性の向上に大きく寄与できることを示す結果が得られた．

4.4　ターゲットマシン上での生成並列化コード実行時間の評価

次に，ターゲットマシン上での実行時間性能に与える影響を評価した．本手法が有用になると考えられる，プログラムを実行するターゲットマシンの CPU 世代が変更し実行プロファイルに変化が生じた，という状況をシミュレートするため，ODROID-XU3 を用いて，周波数を変えたときの 2 コア，4 コア実行時間の計測を行った．使用する周波数としては 200MHz，400MHz，600MHz，800MHz，1GHz，1.2GHz，1.4GHz を採用した．そして，テストプログラム A・B 共に，各コア，各周波数で実行して得られた実行時間を比較した．従来手法の実行時間で正規化したテストプログラム A の実行時間を**図 5** に，テストプログラム B の実行時間を**図 6** にそれぞれ示す．図 5 の通り，テストプログラム A の実行時間比（本手法適用並列化コード実行時間/従来手法適用並列化コード実行時間）は 200MHz と 400MHz 及び 1.4GHz で 1.01，600MHz で 0.99，800MHz で 1.02，1GHz と 1.2GHz で 1.00，となり，誤差は 3%以下となった．また，図 6 の通り，テストプログラム B の実行時間比は 600MHz と 1.2GHz で 0.99，その他の周波数では 1.00 となり，誤差は 1%以下となった．

テストプログラム A・B，各コア，各周波数，で計測された実行時間の比較では誤差が 3%以下となり，周波数やコア，プログラムに関わらず従来手法と本手法でプログラムの実行時間が変わらない，つまり並列化性能が変わらないことを確認できた．

5.　まとめ

本稿では，プロファイル・フィードバックを用いて，自動並列化する際，繰り返し解析・リストラクチャリングを行わず，プロファイル情報からスケジューリングを行い，並列化を行い翻訳時間を短縮する手法を提案した．本手法を用いることで，プログラムサイズが数百万行に及び，翻訳時間に数時間から数十時間かかるプログラムの翻訳時間が大幅に削減されることを示した．例えば Intel Xeon E5-2667 v4 を搭載するマシン上で行った 170 万行のテストプログラム A の翻訳時間の評価では，2 コア用の並列化コード翻訳時間が 208.04[s] から 5.40[s] に削減された．また，同マシン上での 33 万行のテストプログラム B の 2 コア用並列化コード翻訳時間が 6935.19[s] から 1.27[s] に削減された．また，生成された自動並列化コードの並列化性能を評価したところ，ODROID-XU3 ボード上で，従来手法と本手法の実行時間の差が 1%から 3%となり，並列化性能に差異が生じないことが確認できた．よって，実行時プロファイルを複数回取得し，並列化性能を向上させる場面においては，本手法は並列化コード生成のための翻訳時間を大幅に削減することに寄与し，ソフトウェア生産性を向上させることができることが確かめられた．

謝辞　この成果は，国立研究開発法人新エネルギー・産業技術総合開発機構（NEDO）「シード期の研究開発型ベンチャーに対する事業化支援化事業」の委託事業の結果得られた．

参考文献

[1] Blake, G., Dreslinski, R. and Mudge, T.: A survey of multicore processors, IEEE SIGNAL PROCESSING MAGAZINE, No. November, pp. 26-37 (2009).
[2] S.Mehta and P-C, Yew.: Improving Compiler Scalability: Optimizing Large Programs at Small Price, Proceedings of the 36th AM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'15), pp 143-152 (2015)
[3] 笠原博徳：最先端の自動並列化コンパイラ技術，情報処理学会誌，Vol.44 No.4，pp.384-392 (2003).
[4] 本多弘樹，岩田雅彦，笠原博徳：Fortran プログラム粗粒

度タスク間の並列性検出手法，電子情報通信学会論文誌，Vol.J73-D-1，No.12，pp.951-960 (1990).
[5] 小幡元樹，笠原博徳，木村啓二：OSCAR チップマルチプロセッサ上でのマルチグレイン並列処理，情報処理学会研究報告，ARC2002-149-20(SWoPP2002) (2002).
[6] 笠原博徳：並列処理技術，コロナ社 (1991).
[7] 梅田 弾，鈴木 貴広，見神 広紀，木村 啓二，笠原 博徳：組み込み向けモデルベース開発アプリケーションのプロファイル情報を用いたマルチコア用マルチグレイン並列処理，情報処理学会論文誌，Vol.57，No.2，pp.1-12，(2016).
[8] Jixin Han, Rina Fujino, Ryota Tamura, Mamoru Shimaoka, Hiroki Mikami, Moriyuki Takamura, Sachio Kamiya, Kazuhiko Suzuki, Takahiro Miyajima, Keiji Kimura, Hironori Kasahara: Reducing Parallelizing Compilation Time by Removing Redundant Analysis, Systems, Programming, Languages and Applications: Software for Humanity (SPLASH), (2016).
[9] Hardkernel.co.,Ltd: ODROID-XU3, 入手先 ⟨http://www.hardkernel.com/main/products/prdt_info.php?g_code=g140448267127⟩ (2017.02.01).
[10] Keiji Kimura, Gakuho Taguchi, Hironori Kasahara: Accelerating Multicore Architecture Simulation Using Application Profile, 2016 IEEE 10th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC) (2016).
[11] Motoki Obata, Jun Shirako, Hiroki Kaminaga, Kazuhisa Ishizaka, Hironori Kasahara, "Hierarchical Parallelism Control for Multigrain Parallel Processing", Lecture Notes in Computer Science, Springer, Vol. 2481, pp.31-44, (2005).