情報処理学会論文誌 Vol.65 No.2 539–551 (Feb. 2024)

OSCAR自動並列化コンパイラによる並列化オーバヘッド削減のためのタスク融合手法を用いた実ラダーアプリケーションの並列化

川角 冬馬[1,a)] 見神 広紀[1,b)] 吉川 智哉[2,c)] 細見 武郎[2,d)]
追立 真吾[2,e)] 木村 啓二[1,f)] 笠原 博徳[1,g)]

受付日 2023年5月21日, 採録日 2023年11月7日

概要：半導体製造等を含めたファクトリーオートメーション（FA）では，ラダー言語で制御されるプログラマブルロジックコントローラ（PLC）が広く利用されている．製造装置の応答性向上のため，半導体製造等の複雑な工程を扱う大規模プラント制御ではPLCの高速化が求められている．クロック周波数引き上げによる高速化はPLC搭載CPUに要求される高耐久性の観点から適さず，マルチコアCPUによる並列処理が期待される．しかしながら，ラダープログラムの各制御処理は分岐処理が多く発生し，分岐後の計算が小さい．このため，従来の並列化手法では並列化利得よりも同期等の並列化オーバヘッドが大きくなるという問題が存在した．本論文では，OSCARコンパイラによるラダープログラムの並列性抽出および高速化を行う手法を提案する．さらに，ラダープログラムの並列性解析を高速に行う手法を提案する．提案手法ではラダープログラムを，配列アクセス範囲情報付きの等価なCプログラムへ変換する処理系を開発し，その出力を新規のタスク融合手法を実装したOSCARコンパイラによって解析した．提案手法の有効性を，実マルチコアボード上で評価したところ，産業界提供の大規模実FAラダープログラムに対し，ARM 2コア上で1.4倍の速度向上が得られることを確認できた．また，コンパイル時間については配列アクセス範囲情報付与により最大で8.4倍高速化できることが確認できた．

キーワード：並列処理，自動並列化コンパイラ，ラダープログラム

Parallelizing Ladder Applications with Task Fusion Techniques for Reducing Parallelization Overhead by OSCAR Automatic Parallelizing Compiler

Tohma Kawasumi[1,a)] Hiroki Mikami[1,b)] Tomoya Yoshikawa[2,c)] Takero Hosomi[2,d)]
Shingo Oidate[2,e)] Keiji Kimura[1,f)] Hironori Kasahara[1,g)]

Received: May 21, 2023, Accepted: November 7, 2023

Abstract: Programmable Logic Controllers (PLC) operated by ladder programs have been widely used for factory automation. Large-scale plant control, such as semiconductor manufacturing, requires fast PLCs because their complicated manufacturing processes require higher response time. For this purpose, CPU clock frequency improvement is unsuitable since PLCs require high durability. Therefore, parallelizing ladder programs is a promising approach for improving the performance of PLCs. However, due to the small task cost in a ladder program, the parallelization overhead becomes a critical issue in conventional parallelization methods. This paper proposes a method for exploiting parallelism from ladder programs and speeding them up by the OSCAR compiler. Besides, we propose a highly efficient method for exploiting parallelism from ladder programs. We developed an automatic translator to derive a C program with array access range information from ladder programs. The OSCAR compiler then takes the output C program to parallelize it by utilizing the newly proposed task fusion method. Performance evaluation using three real ladder applications and an ARM multi-core board confirmed that our proposal improves the performance by up to 1.4 times on two cores. Also, we confirmed that our proposal improves the compilation time by up to 8.4 times.

Keywords: parallel processing, parallelizing compilers, ladder program

1. はじめに

工場の自動化にはPLCが広く用いられている [1], [2], [3]. 従来のシーケンス制御で用いられてきた電磁リレーと比較してPLCを用いたシーケンス制御では制御内容の変更が配線の変更ではなくソフトウェアの変更によって行えるため保守性の面で優れている．また，PLCは一般のPCと比較して耐久性も優れており工場のようなノイズの多い環境でも高い信頼性を確保できる [4].

特に半導体製造のような複雑な工程を持つ大規模なプラントの制御では，高速・高応答なPLCが要求される [5], [6], [7], [8]．PLCを高速化することでPLCがセンサ入力を検知するまでの時間が短縮され，PLCの応答性が向上する [9]．また，高速なPLCにはより多くのセンサが接続可能であるため，PLC台数削減による保守性の向上が期待できる.

しかしながら，PLCに搭載されているCPUは工場内の電磁波等によるノイズにさらされた環境下においても高い信頼性を確保する必要があり，高周波数化による高速化は適さない．そのため，PLCへ入力されるラダー言語で記述されたプログラムの並列化が，耐久性を求められるPLCを高速化する手法として有望なアプローチとなる.

ラダープログラムの高速化に関連した先行研究としてラダープログラムの実行条件判定の命令数を削減して高速化する手法 [10], [11] やラダープログラムを一連の命令ブロック単位で並列化する手法 [12] が提案されてきた．しかし，これらの研究ではいずれも実マルチコアボード上での評価は行われていなかった.

本論文ではラダープログラムを同等のCプログラムへ変換し，それをOSCAR自動並列化コンパイラ [13] へ入力して並列性解析を行って高速化する手法を提案する．ラダープログラムの解析にあたり，ラダープログラムをParallelizable C規約 [14] に準拠した並列性解析が容易な形式の等価なCプログラムへ変換する処理系を開発した．提案する処理系によりラダープログラムから変換されたCプログラムはOSCARコンパイラによる並列性解析に適した形式ではあるが，一方でプログラム中に存在する大量の配列の定義・参照情報収集処理がコンパイル時間増大を引き起こす．そのため，本処理系の出力するCプログラムには，上記解析処理時間を削減するために並列性解析のための配列定義・参照情報を埋め込むことができる．これを用いることで高速に並列性解析を行うことが可能となる．この処理系から得られたCプログラムを，新規のタスク融合機能を実装したOSCAR自動並列化コンパイラを用いて並列性を抽出し，並列Cコードを生成した．本論文で提案する新規のタスク融合機能は，ラダープログラムの初期化タスク群とデータ書き出しタスク群が各々で並列実行可能かつタスクコストが小さいという特徴に着目している．また，タスク融合によるプログラム並列性の低下を抑えながら並列化オーバヘッドを削減するために融合対象を想定実行環境における同期コスト以下の小タスクに限定し，融合のグルーピングを関連タスクどうしとした．さらに，得られた並列コードを実マルチコアボード上で評価を行った．以上をまとめると，本論文の貢献は以下のとおりとなる：

- ラダープログラムを，Parallelizable C規約 [14] に準拠した並列性解析の容易なCプログラムへの変換手法の提案
- Cプログラムに変換したラダープログラムの自動並列化コンパイル時間削減手法の提案
- 並列化オーバヘッド削減のためのプログラムの持つ並列度と同期等並列処理オーバヘッドの両方を考慮したマクロタスク融合手法の提案
- 実マルチコアボード上における提案手法の実行性能評価

本論文は著者等がETNET2022にて発表した研究 [15] およびLCPC2022で発表した研究 [16] に基づいている．研究 [15] に対し，本論文では5章で述べるタスク融合手法，4.4.3項で提案するCプログラムに変換したラダープログラムのコンパイル時間削減手法，および実マルチコアボード上における提案手法の実行性能評価を新たに実施した．さらに，研究 [16] に対し，本論文ではコンパイル時間削減手法の新たな提案，および実マルチコアボード上における性能評価に対し性質の異なる評価プログラム追加による拡充を実施した.

以下，本論文では，2章で本論文で高速化を行うラダー言語について説明する．3章ではOSCAR自動並列化コンパイラの概要を説明し，4章ではラダープログラムをCプログラムへ変換する処理系について説明する．5章でラダープログラム高速化のためのタスク融合手法を説明し，6章ではその評価結果を述べる．7章で関連研究を述べた後，8章でまとめる.

2. ラダー言語

本章では本論文で解析対象として扱うラダー言語の構造や命令について概説する.

2.1 ラダープログラムの概要

ラダープログラムは従来の電磁リレーによるシーケンス

[1] 早稲田大学
Waseda University, Shinjuku, Tokyo 169–8555, Japan
[2] 三菱電機株式会社
Mitsubishi Electric, Chiyoda, Tokyo 100–8310, Japan
a) tohmak@asagi.waseda.jp
b) hiroki@kasahara.cs.waseda.ac.jp
c) Yoshikawa.Tomoya@aj.mitsubishielectric.co.jp
d) Hosomi.Takero@ap.mitsubishielectric.co.jp
e) Oidate.Shingo@dx.mitsubishielectric.co.jp
f) keiji@waseda.jp
g) kasahara@waseda.jp

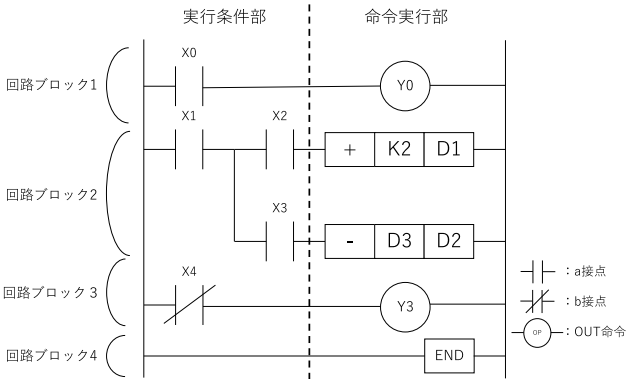


図 1 ラダー図の例
Fig. 1 An example of a Ladder diagram.

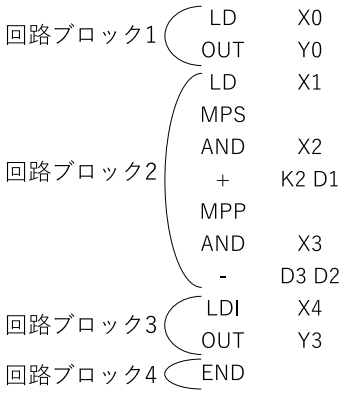


図 2 図 1 に対応するラダーインストラクションリストの例
Fig. 2 An example of a Ladder instruction list for Fig. 1.

制御をモデル化したものである．ラダープログラムの表現方法には制御回路を図で表したラダー図形式と命令列を文字で表現したインストラクションリスト（IL）方式が主に存在する．プログラム開発者は一般にラダー図形式でプログラミングを行うが，ラダープログラムを IL 形式で出力し他のツールへ入力することも可能である．

2.2 ラダープログラムの構成

ラダー図の例を**図 1** に示す．ラダープログラムは左右の母線と，それらをつなぐ複数の回路で構成される．回路は左側の実行条件部と右側の命令実行部で構成されている．さらに，回路の左側の母線との接点から，その接点とつながっている一番右下の母線との接点までの一連の回路を回路ブロックと呼ぶ．

回路ブロックに含まれる各命令はオペランドとしてデバイスの種別とデバイス番号を持つ．代表的なデバイスとしては入力であるデバイス X，出力であるデバイス Y，各々がデバイス番号の即値を持つデバイス K や 16 ビットワードデータを保持できるデバイス D 等が利用できる．さらにオフセットデバイスと呼ばれる，他のデバイスの番号を修飾し，オフセット付きアクセスを実現するデバイス Z も利用可能である．

ラダープログラムの実行はラダー図の左から右，上から下の順番である．ラダー図終端の END 命令が実行されると，プログラムが再びラダー図左上から再開される．このラダープログラム全体実行の 1 周をスキャンと呼び，スキャンにかかる時間をスキャンタイムと呼ぶ．各スキャンはループ誘導変数によってイテレートされず，また一部の命令は前回スキャンにおけるデバイスの状態との差分をとるため，スキャンのループそのものは並列化に適さない．

ラダー図の実行条件部は a 接点と b 接点の組合せによって構成される．a 接点は自身が保持するオペランドが 1 のときに導通する．b 接点は a 接点とは逆にオペランドが 0 のときに導通する．

ラダー図の命令実行部は加減算等の通常の命令と OUT 命令で構成される．OUT 命令は他の命令と異なり実行条件部が導通しない場合でも実行される命令であり，導通している場合はオペランドのデバイスに 1 を格納し，導通していない場合はオペランドへ 0 を格納する．

例として，図 1 で示した回路の動作を説明する．図の回路は上の回路ブロックから順に実行される．回路ブロック 1 はデバイス X0 が 1 のとき Y0 へ 1 を格納し，デバイス X0 が 0 のとき Y0 へ 0 を格納する．回路ブロック 2 はデバイス X1 と X2 が 1 のときに加算命令が実行され，X1 と X3 が 1 のときに減算命令が実行される．回路ブロック 3 はデバイス X4 が 0 のとき Y3 へ 1 を格納し，デバイス X4 が 1 のとき Y3 へ 0 を格納する．

2.3 ラダープログラムの命令

図 2 に IL の例を示す．この IL は図 1 のラダープログラムに対応している．たとえば，図 2 の最初の LD 命令と OUT 命令は図 1 の回路ブロック 1 に対応している．ラダープログラムの IL では，回路ブロックの実行条件部と左側母線との接続を LD 命令または LDI 命令を用いて表現する．LD 命令は a 接点と母線との接続を表し，LDI 命令は b 接点と母線との接続を表す．また各接点どうしの接続には直列接続を示す AND 命令と並列接続を示す OR 命令を用いる．さらに，1 度 LD 命令等で生成した接点演算の結果を複数の箇所で再利用するために MPS，MRD，MPP 命令を用いることもできる．MPS 命令によって接点演算結果をスタックへプッシュし，MRD 命令によりスタックロードを行い，MPP 命令によってスタックのポップを行える．これらを用いることで，複雑な実行条件を複数の実行命令に対して容易に流用することが可能となる．

3. OSCAR 自動並列化コンパイラ

本章では本論文の評価で用いた OSCAR 自動並列化コンパイラについて述べる．

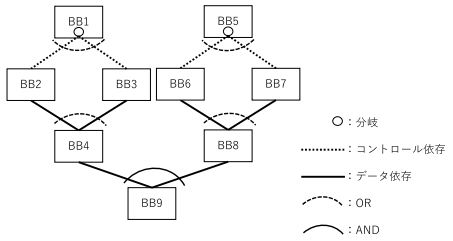


図 3 MTG の例
Fig. 3 An example of MTG.

3.1 粗粒度タスク並列処理

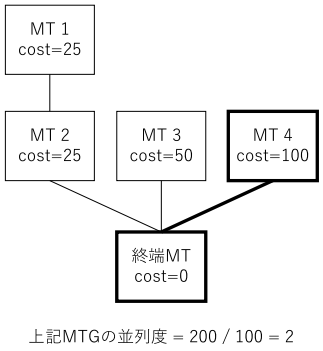
OSCAR 自動並列化コンパイラは従来の並列コンパイラが行ってきたループの並列性抽出に加えて，プログラムのタスク間の並列性を抽出することができる．粗粒度タスク並列化を行う際，OSCAR 自動並列化コンパイラはソースプログラムを基本ブロック（BB），繰返しブロック（RB），サブルーチンブロック（SB）の 3 種のマクロタスク（MT）に分割する．BB は途中で他のブロックへ制御が移らない連続した命令列を含むブロック，RB はループを含むブロック，SB は関数呼び出しを含むブロックをそれぞれ表している．タスク分割の後，各 MT のデータ依存とコントロールフローを解析して MT をノード，MT 間データ依存とコントロールフローをエッジとしたマクロフローグラフ（MFG）を生成する．さらに，MFG の各タスクに対して各々のデータ依存とコントロールフローを用いた最早実行可能条件解析を行い，マクロタスクグラフ（MTG）を生成する [17]．MTG の例を**図 3** に示す．図の各ノードは MT を表しており，点線エッジはコントロール依存，実線エッジはデータ依存関係をそれぞれ表している．また，MT 下部の円は分岐の始点を表す．さらに，点線の円弧は複数エッジの OR 関係を表し，実線の円弧は複数エッジの AND 関係をそれぞれ表す．たとえば，図の MTG は BB1 が BB2 または BB3 へ分岐することを示している．また，BB9 は BB4 の終了かつ BB8 の終了によって計算に必要なデータが充足され，実行可能になることが示されている．

MTG の生成後，OSCAR コンパイラは各 MT のコストおよび MTG の並列度を計算する [18]．各 MT のコストは，MT 内部の OSCAR 中間表現の命令の重みの値の総和となる．各命令の重みの値はターゲットとなるアーキテクチャによって変化する．並列度は以下の式で計算される．

$$Para = Seq/CP \tag{1}$$

ここで，Para は並列度，Seq はプログラムの逐次実行コスト，CP は MTG の最長パス長をそれぞれ表している．この式で計算される並列度は並列化オーバヘッドのない理想的な並列度となる．**図 4** に並列度の計算例を示す．図中の太線部分は MTG の最長パスを表す．上述の式にあてはめると図の Para は以下のように計算できる．

$$Para = (25 + 25 + 50 + 100)/100 = 2 \tag{2}$$



上記MTGの並列度 = 200 / 100 = 2

図 4 MTG の並列度計算
Fig. 4 Calculation of MTG parallelism.

この計算により，図の MTG の元になったプログラムは最大で 2 倍の速度向上が可能であることが分かる．

MTG の生成後，MT が各プロセッサコアへスケジューリングされることでプログラムが並列実行される．プログラムの条件分岐によって実行時非決定性が存在する場合は実行時に MT をコアへ割り当てるダイナミックスケジューリングが適用される．実行時非決定性がプログラムに存在しない場合はコンパイル時に MT をコアへ割り当てるスタティックスケジューリングが適用される．

3.2 制御系アプリケーション高速化のためのタスク融合

本節では，従来の OSCAR 自動並列化コンパイラが行ってきた最適化手法のうち，ラダープログラムのような制御系アプリケーションに対して有効な高速化手法について述べる．

3.2.1 スタティックスケジューリング適用のためのタスク融合

ダイナミックスケジューリングはプログラム実行時にコアに対してタスクを割り当てるため，スケジューリング時にオーバヘッドが発生する．このオーバヘッドを解消するため OSCAR 自動並列化コンパイラでは実行時非決定性を MT 内部へ隠蔽するタスク融合を行う [19]．具体的には条件分岐を判定する MT と，その MT とコントロールフローがつながっている MT を 1 つの MT と見なして単一のコアへ割り当てる．**図 5** に図 3 の MTG にタスク融合を適用した場合の融合結果を示す．図 5 の点線枠は条件分岐の始端から終端までを一括りにした部分 MTG を表している．この部分 MTG ををを 1MT と見なすことで実行時非決定性がなくなり，スタティックスケジューリングが可能となる．以降の項では，条件分岐はすべて MT 内部に隠蔽されているものとする．

3.2.2 同期オーバヘッド削減のためのタスク融合

OSCAR 自動並列化コンパイラは MT 間の同期オーバヘッドやデータ転送オーバヘッドを削減するため以下の 2 種類のタスク融合を行う [20].

(1) MTG 内のある MT A と MT B が 1 つのデータ依存

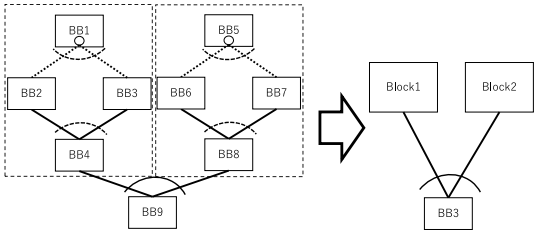


図 5 スタティックスケジューリング適用のためのマクロタスク融合の例

Fig. 5 An example of macro task fusion for applying static scheduling.

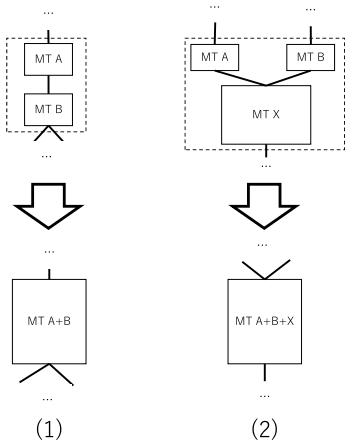


図 6 並列化オーバヘッド削減のためのマクロタスク融合

Fig. 6 Macro task fusion for reducing parallel overheads.

エッジのみでつながっている場合，それら 2 つの MT を融合．

(2) MTG 内の互いに依存関係のない MT A と MT B が別の 1 つの MT X を共通の後続 MT として持っており，なおかつ同期コストを含めた先行 2MT の並列実行コストよりも 2MT のコストの総和が小さい場合，3MT を融合．

図 6 に上記 2 つの MT 融合の図を示す．これらのタスク融合により，データ依存によって並列実行のできない MT でのデータ転送オーバヘッドを削減できる．また，MT のコストが小さく，並列化オーバヘッドが並列化による利得を上回ってしまうような並列化も抑制できる．

4. ラダープログラムの C 言語変換手法

本章では OSCAR 自動並列化コンパイラによってラダープログラムの並列性を抽出するための，ラダープログラムを C プログラムへ変換する処理系について述べる．本研究で出力する C コードは，自動並列化コンパイラによる並列性抽出をサポートするプログラム記述規約である Parallelizable C [14] に準拠している．

4.1 デバイスの表現方法

ラダープログラムの各命令のオペランドとして用いられるデバイスは C 言語の配列へ変換した．デバイス種別を配列名，デバイス番号を配列添字とすることによって各命令間でのデータ依存の解析が容易に行える．ただし，2 章で述べたデバイス K は各々がデバイス番号の即値を持つため，配列ではなくデバイス番号単体へ変換した．

```
LD    X1                if (X[0x1]) {
MPS                       if (X[0x2]){
AND   X2                     D[1] = D[1] + 2;
+     K2 D1                  }
MPP                       if (X[0x3]) {
AND   X3                     D[2] = D[2] – D[3];
-     D3 D2                  }
                          }
```

図 7 MPS，MPP 命令の C 変換例

Fig. 7 An example of MPS and MPP translation.

4.2 実行条件部の変換

2.3 節で述べた MPS 命令等を含まない単純な実行条件部は接点演算のオペランドを条件式に持つ if 節へ変換した．前述のとおり，ラダーには接点演算結果をスタックへプッシュする MPS 命令とスタックトップをロードする MRD 命令とスタックのポップを行う MPP 命令が存在する．ラダーの IL はこれらの命令を用いることで複数箇所で接点演算の結果を流用しており，コード量を削減している．たとえば，図 1 の回路ブロック 2 では LD X1 の結果を MPS 命令と MPP 命令で再利用することで 2 つの実行命令に対して 1 つの a 接点を接続している．本研究で開発した変換処理系では MPS，MRD，MPP 命令によるコントロールフローを C 言語の if 節のネスト構造へ変換することでラダーと同様の条件分岐判定の再利用を行う．変換処理系では 3 命令を以下のとおり扱う．

(1) MPS：直前までの接点演算結果を用いて if 節を生成し，以降の変換結果をこの if 節内部へ出力する．

(2) MRD：直前までの接点演算結果を用いて if 節を出力．

(3) MPP：直前までの接点演算結果を用いて if 節を出力．その直後の命令実行部の出力完了後，if 節を 1 つ閉じる．

図 1 で示した回路ブロック 2 を変換した例を**図 7** に示す．まず，最初の LD 命令と MPS 命令によってデバイス X1 の if 節が生成される．続く AND 命令と+命令によって X1 の if 節内部に X2 の if 節と加算文が出力される．さらに，MPP 命令，AND 命令，-命令によって X3 の if 節とその内部の減算命令が出力され，最外側 X1 の if 節が閉じられる．このように if 節のネスト構造を用いることで X1 の条件分岐の出力を 1 回のみにしている．ネスト構造を用いない場合，各実行命令に対して X1 の冗長な条件分岐が生成されてしまう．

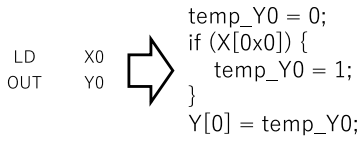


図 8 OUT 命令の変換
Fig. 8 Translation of OUT instruction.

4.3 命令実行部の変換

ラダーの命令実行部の各命令は図 7 で示したように同等の C 言語を用いて定義した．その中で特殊な扱いが必要になる OUT 命令についてここでは詳しく説明する．

ラダーの OUT 命令は 2.2 節で示したとおり，実行条件が満たされた場合はオペランドを 1 に設定し，実行条件が満たされない場合はオペランドを 0 に設定する命令である．そこで，本研究で開発した変換処理系では**図 8** に示すような C 言語のコードに OUT 命令を変換した．この図は図 1 中の回路ブロック 1 を変換する例となっている．変換処理系が OUT 命令を変換する際はまず，OUT 命令を含む回路ブロックの直前で一時変数をゼロ初期化する．その後，OUT 命令が実行される箇所に一時変数への 1 代入を配置する．最後に回路ブロック直後のところに一時変数の内容をデバイス本体へ格納する文を書くことで OUT 命令の変換が完了する．この変換によって OUT 命令の実行条件が満たされた場合は，一時変数を介してデバイスに 1 が代入される．逆に実行条件が満たされない場合は，ゼロ初期化されたままの一時変数を介してデバイスへ 0 が代入される．

4.4 OSCAR コンパイラによる並列性抽出に向けた変換

本節では，ラダープログラムから C プログラムへの変換に際して，OSCAR コンパイラによる並列性抽出を補助するコード生成手法について述べる．

4.4.1 オフセットデバイスの変換

オフセットデバイス Z はスカラ変数へ変換した．さらに，デバイスのオフセットアクセスは配列添字の加算を用いて表現した．たとえば，D10Z5 のようなデバイスは配列 D とスカラ変数 Z5 を用いて D[10 + Z5] へ変換される．この変換において配列は線形添字式のみを持つため，OSCAR コンパイラによるデータ依存解析が容易となる．

4.4.2 可変長引数を受け取る命令の変換

ラダープログラムの命令には引数の個数が可変となっている命令が存在する．たとえばラダーの加算命令は引数を 2 個から 28 個までの範囲で受け取ることが可能となっている．しかし，Parallelizable C [14] では C 言語の可変長引数は推奨されていない．そのため，本研究で開発した変換処理系ではこれらの命令定義を引数の個数によって分けることにより可変長引数を用いない C コードを生成している．

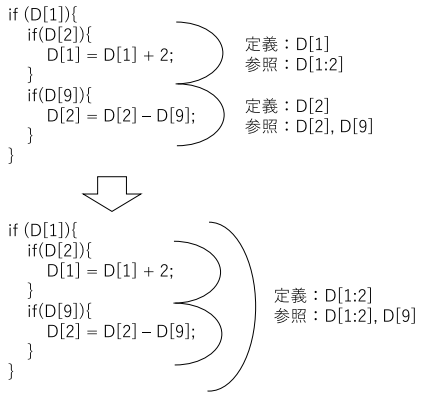


図 9 配列アクセス範囲解析の例
Fig. 9 An example of array access range analysis.

4.4.3 回路ブロックへの配列定義・参照範囲情報の付与

ラダープログラム上でデバイスが定義・参照される場合，前述のオフセットアクセスを除けば，ほとんどの場合はアクセス先デバイス番号に定数値のみが用いられる．4.1 節で述べたとおり，これらのデバイスは変換後の C プログラムは配列として表現され，ラダープログラムの並列性抽出はこれら配列添え字に基づいた配列間の依存解析により行われる．すなわち，OSCAR コンパイラは，粗粒度並列処理の単位である MT の依存解析に用いる情報として，MT 内部に含まれる各命令のすべての配列添え字を集約して対象 MT 全体の配列アクセス範囲を算出する．ここで，配列範囲情報の単位は，同一配列中の連続した配列要素の範囲である．

図 9 に 1 回路ブロックの配列アクセス範囲解析の例を示す．まず，回路ブロック内部に含まれる各ブロックに対し，配列範囲重複の確認と集約を行いながら配列定義参照情報を算出する．たとえば，図 9 中の回路ブロック下部のブロックにはデバイス D の 9 番への参照が 2 つ存在するが，生成する範囲情報は 1 つのみとしている．その後，同様に範囲重複の確認を行いながらすべてのブロックの配列アクセス範囲を集約し，MT 全体の配列アクセス範囲情報を算出する．MT が複数のネストされたブロックから構成される場合，通常のリージョンに基づくデータフロー解析 [21] と同様に，内側ブロックから外側ブロックに向けて範囲情報の集約処理が行われる．

各 MT の依存解析完了後，MT 間の依存解析を，これら配列アクセス範囲に基づき行う．4.4.1 項で述べたとおり，配列添え字は線形添え字となっており，変換後の配列アクセス範囲情報のコンパイラによる自動生成は十分可能である．

しかしながら，ラダーのほぼすべての命令にデバイスが出現することから，大規模プラントのラダープログラムの並列化ではラダーステップ数と用いられるデバイス数が膨大となり，配列アクセス範囲情報作成およびその中の配列アクセス範囲統合処理が並列化コンパイル時のボトルネッ

クとなってしまう．6 章で評価するプログラムの解析においては，コンパイル時間の 90%が配列アクセス範囲情報の作成に費やされる．

そこで，本研究で開発した変換処理系では変換結果の C コードの各回路ブロックに対して配列の定義・参照範囲をコンパイラ指示文として埋め込んでいる．この付与情報を参照することで，OSCAR コンパイラは中間表現から定数値の配列アクセス範囲情報を生成する必要がなくなるため，コンパイル時間が削減される．本手法によるコンパイル時間削減効果については後述の性能評価の節（6.3 節）で述べる．

4.5 変換対象外の命令

本研究で開発した変換処理系において立上り・立下り命令は変換対象外となっている．立上り命令とは実行条件となるデバイスが前回スキャンと比較して 0 から 1 へ変化している際に実行される命令を指す．立下り命令は，逆に実行条件となるデバイスが前回スキャンと比較して 1 から 0 へ変化している際に実行される命令を指す．これらの命令は各スキャン間での依存解析を行わない本研究のスコープの対象外であるため，変換は行わない．

5. 並列化オーバヘッド削減のためのタスク融合手法

本章ではラダープログラムの並列実行時に速度鈍化の原因となる同期・データ転送オーバヘッドの削減手法について述べる．

5.1 従来提案されている自動車エンジン制御用の並列処理手法によるラダープログラム解析

ラダー言語によって記述されるシーケンス制御プログラムに近いリアルタイム制御プログラムの並列化手法として，3.2 節で述べた自動車エンジン制御プログラムの並列処理 [19] が提案されている．この手法を用いてラダープログラムの並列性解析を行うと**図 10** に示す MTG が生成される．図のノードは MT，エッジはデータ依存をそれぞれ表している．この MTG の並列度は 2.5 であるため，解析結果のうえでは 2 コアを用いた並列化によって高速化が見込める．しかしながら，MTG の平均タスクコストは ARM CPU を搭載した実マルチコアボードの同期コストと同程度であるため実際には速度向上しない．本論文の提案手法では MTG の平均タスクコストを増加させることにより，ラダープログラムのようなタスクコストの小さいプログラムの並列実行性能向上を行う．

5.2 提案手法

まず最適化の対象とする，タスクコストの小さい小 MT を定義する．本手法における小 MT とは，想定する実行環境における同期コスト以下の実行コストを持つ MT を指す．上述の小 MT に対して以下の 4 種類の MT 融合を適用する．

(1) 先行タスクとして MT X を唯一持つ小 MT Y について，X の末尾に Y を融合する．
(2) 後続タスクとして MT X を唯一持つ小 MT Y について，X の先頭に Y を融合する．
(3) 先行タスクを 1 つも持たず，なおかつ共通の後続タスクを持つ小 MT 群について，それらを 1MT に融合する．
(4) 後続タスクを 1 つも持たず，なおかつ共通の先行タスクを持つ小 MT 群について，それらを 1MT に融合する．

各融合の概略図を**図 11** と**図 12** に示す．図の各点線枠は MTG の融合範囲を表している．図 11 の (1) では小 MT Y が先行タスクとして MT X のみを持っているため，MT X と MT Y を 1 タスクに融合している．図 11 の (2) では小 MT Y が後続タスクとして MT X のみを持っているため，MT Y と MT X を 1 タスクに融合している．図 12 の

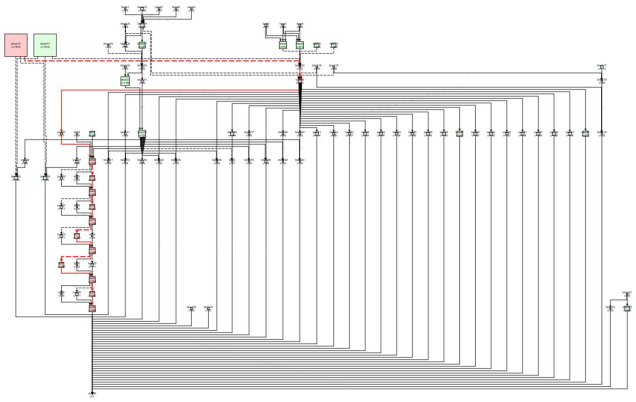


図 10 提案手法を用いない場合の評価用ラダープログラム 1 の MTG [16]
Fig. 10 MTG of an evaluation Ladder program1 without the proposed method [16].

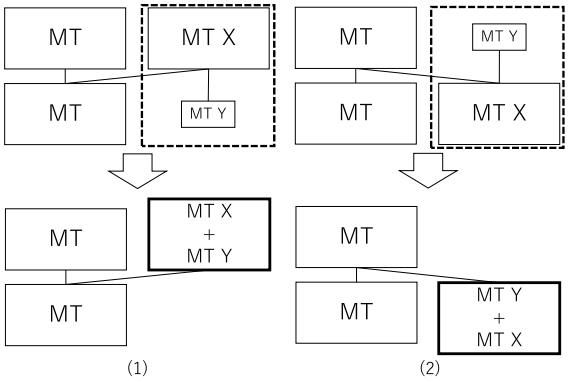


図 11 並列化オーバヘッド削減のための小 MT 融合 (1)，(2) [16]
Fig. 11 MT fusion methods (1) and (2) for reducing parallelization overhead [16].

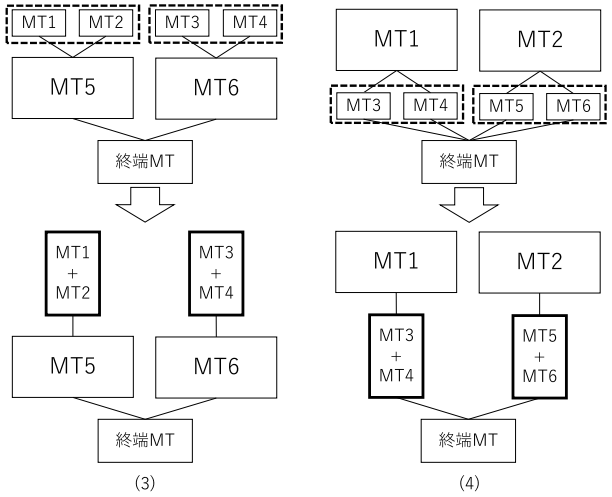


図 12 並列化オーバヘッド削減のための小 MT 融合 (3)，(4) [16]
Fig. 12 MT fusion methods (3) and (4) for reducing parallelization overhead [16].

(3) では共通の後続タスク MT5 を持ち，なおかつ先行タスクを持たない小 MT1 と小 MT2 を 1 タスクに融合している．同様に MT3 と MT4 も融合する．図 12 の (4) では共通の先行タスク MT1 を持ち，なおかつ後続タスクを持たない小 MT3 と小 MT4 を 1 タスクに融合している．同様に MT5 と MT6 も融合する．図 11 の (1)，(2) によって 3.2.2 項で述べた MT 融合では対処しきれなかった，並列化による高速化よりもデータ転送オーバヘッドの方が大きくなる箇所の最適化が可能となった．図 12 の (3)，(4) ではそれぞれラダープログラムに頻出するデバイスの初期化を行う小 MT 群と外部デバイスへの結果書き出しを行う小 MT 群を融合しタスクコストを増大させ，相対的な並列化オーバヘッドを削減することができる．また，融合対象 MT を共通の関係タスクがある MT どうしとすることで負荷不均衡や並列性の低下を抑制しながらタスクコストを増大させている．

6. 実ラダーアプリケーションを用いた提案手法の性能評価

本章では産業界から提供された実ラダーアプリケーションを用いた提案手法の性能評価について述べる．

6.1 評価方法

評価用アプリケーションとして，産業界から提供された 3 つの工場自動化用実ラダーアプリケーションを使用した．**表 1** に評価に用いたプログラムの規模と逐次実行クロック数を示す．表 1 のプログラム 1 が，研究 [16] において用いたプログラムであり，プログラム 2 とプログラム 3 が本論文で追加したプログラムである．本論文で追加したプログラム 2 と 3 はサブルーチンコールを行う命令が複数回実行され，1 回のサブルーチンコールでは複数の処理が行われる．そのため，1 ステップあたりの計算量が大きい．一方でプログラム 1 はサブルーチンコールを含まないため 1 ステップあたりの計算量がプログラム 2，3 と比べて小さい．以上の理由によりプログラム 1 とプログラム 2，3 はラダーステップ数では同等であるが，実行にかかるクロック数には約 9 倍の差がある．またプログラム 2 と 3 に関しては，プログラム 2 ではラダー回路中の繰り返し使用されるブロックを部品化するファンクションブロックを用いてラダープログラムが記述されているが，プログラム 3 ではファンクションブロックを用いていないことが，プログラム構成上の大きな違いとなる．

表 1 評価プログラムの概要
Table 1 Summary of the evaluated programs.

	ラダーステップ数	ZCU102 上での C 変換後の 逐次実行クロック数
プログラム 1	1,766	1,614
プログラム 2	1,872	15,232
プログラム 3	1,896	14,864

本論文の性能評価を次の 4 項目を用いて行った．1 点目は提案手法適用有無での平均タスクコストを測定し，提案手法によるタスクコストの増加量を確認した．2 点目は手法適用有無での並列度測定および同期コード数の確認である．これらによって提案手法による並列度の低下量と並列化オーバヘッドを確認した．3 点目としてラダー変換処理系から得られた C プログラムを実マルチコアボード上で実行し，実行性能の向上を確認した．2 章で述べたとおりラダープログラムは繰り返し実行される．このため，本評価では変換後のラダープログラムを 10,000 回連続実行し，その平均クロック数を測定した．4 点目として 4.4.3 項で述べた付与情報の有無による並列性解析時間の差を測定した．

6.2 評価環境

評価環境として Xilinx ZCU102 ボード [22] を使用した．本評価ボードは CPU に 4 コアの Cortex-A53，主メモリに DDR4 を 4 GB 搭載している．また本評価では実際の PLC の実行環境に近づけるため CPU の周波数を 300 MHz としている．本性能評価では OS に Ubuntu 20.04.2LTS，ネイティブコンパイラに gcc9.3.0 を用いた．さらに，並列化 API として OpenMP から，各コア用スレッド生成用に “parallel sections”，およびスレッド間メモリビューの一貫性保証に “flush” の各指示文を用いた [23]．なお，本評価環境における同期コストは 200 とし，5.2 節で述べた最適化対象の小 MT はコスト 200 以下の MT とした．このコスト値はビジーウェイトループによるフラグ変数の授受のみを行う OpenMP2 並列 C プログラムを用いて，ZCU102 ボード上で実測した同期に要するクロック数が 180 クロックであったため，200 と設定した．

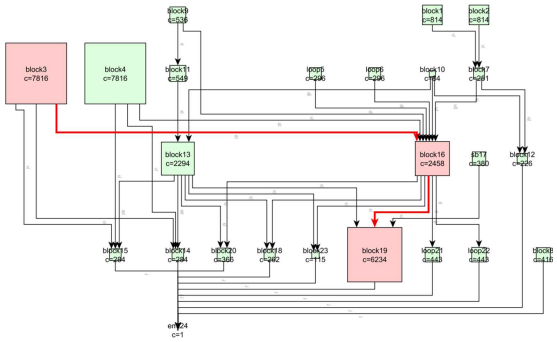


図 13 プログラム 1 の MTG [16]
Fig. 13 The MTG of Program1 [16].

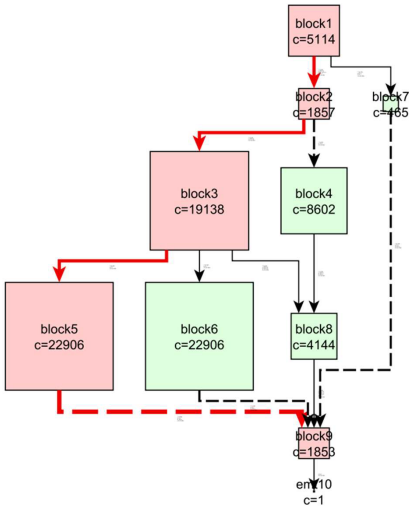


図 14 プログラム 2 の MTG
Fig. 14 The MTG of Program2.

OSCAR コンパイラの実行環境として Ubuntu 22.04.2 LTS と Xeon Gold 6326 CPU（動作周波数 2.9 GHz）と DDR4 256 GB の主メモリを搭載したマシンを用いた．

6.3 評価結果

各プログラムの MTG を**図 13**，**図 14**，**図 15** にそれぞれ示す．これらはすべて本論文の提案手法を適用した場合の MTG となっている．**表 2** に各プログラムの提案手法有無での各 MTG の平均タスクコストの変化，**表 3** に並列コードに含まれる同期コード数の変化，**表 4** に各 MTG の並列度の変化，**図 16** に実マルチコアボード上で変換後のプログラムを 10,000 回連続実行した際の平均の実行クロック数をそれぞれ示す．図 16 において，左の白い棒が C 変換済みラダープログラムを ZCU102 上で逐次実行した場合の実行クロック数，中央の縦縞模様の棒が変換済みプログラムに対して 3 章で示した粗粒度並列化手法およびタスク融合手法を適用した場合の並列実行クロック数，右の黒い棒が 3 章で述べた手法に加えて 5 章で述べた本論文での提案手法を適用して並列実行した場合の並列実行クロック数をそれぞれ表している．逐次実行クロック数測定に用いた

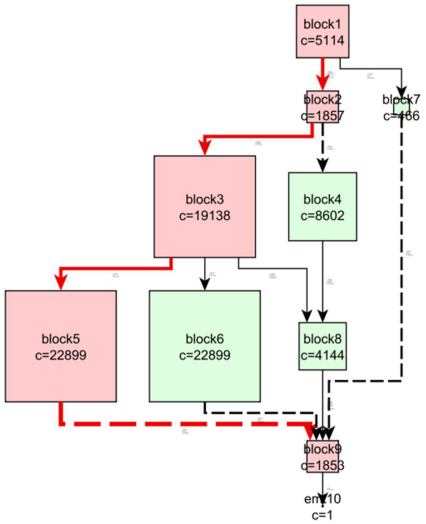


図 15 プログラム 3 の MTG
Fig. 15 The MTG of Program3.

表 2 コンパイラが算出した評価プログラムの平均タスクコスト
Table 2 Average task cost of evaluation programs estimated in the compiler.

	タスクコスト平均（提案手法未適用）	タスクコスト平均（提案手法適用）
プログラム 1	236	1,394
プログラム 2	2,289	8,698
プログラム 3	2,288	8,697

表 3 コンパイラが生成した並列コードに含まれる同期の回数
Table 3 Number of synchronization codes in the evaluation programs generated by the compiler.

	同期回数（提案手法未適用）	同期回数（提案手法適用）
プログラム 1	17	4
プログラム 2	11	2
プログラム 3	11	2

表 4 コンパイラが算出した評価プログラムの並列度
Table 4 Parallelism of evaluation programs estimated in the compiler.

	並列度（提案手法未適用）	並列度（提案手法適用）
プログラム 1	2.5	2.0
プログラム 2	1.7	1.7
プログラム 3	1.7	1.7

プログラムには並列化 API は含まれていない．**表 5** にコンパイル時間の評価結果を示す．表の数値は OSCAR コンパイラ単体の実行秒数を示しており，括弧内の数値は変換処理系の実行秒数を示している．また，左側が 4.4.3 項で述べた手法を用いない場合，右側が 4.4.3 項で述べた手法を用いた場合をそれぞれ表している．

表 2 より，本論文で提案した MT 融合手法によって各評価プログラムの MT のコスト平均はプログラム 1 で 5.9

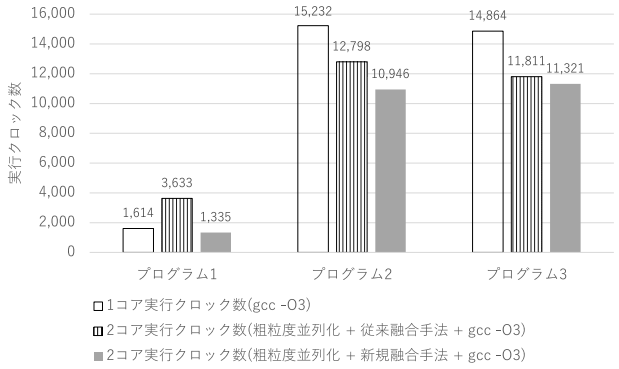


図 16 ZCU102 上での評価用プログラムの並列実行結果
Fig. 16 Parallel execution results of the evaluation programs on ZCU102.

表 5 評価プログラムのコンパイル時間（括弧内はコード変換時間）
Table 5 Compilation time of the evaluation programs (Code generation time is shown in parentheses).

	コンパイル時間 [sec]（変換処理系による配列範囲情報なし）	コンパイル時間 [sec]（変換処理系による配列範囲情報あり）
プログラム 1	10.88 (1.00)	5.03 (1.17)
プログラム 2	12.17 (0.91)	1.45 (1.17)
プログラム 3	8.64 (0.83)	1.32 (1.24)

倍，プログラム 2 で 3.8 倍，プログラム 3 で 3.8 倍に増大することが確認できる．また，表 3 と表 4 よりプログラム 1 以外では MTG の並列度を損なうことなく並列コードの同期回数を削減できていることが確認できた．さらに，図 16 より実マルチコアボード上において，逐次実行と比較し，プログラム 1 は 2 コアで 1.2 倍，プログラム 2 は 2 コアで 1.4 倍，プログラム 3 は 2 コアで 1.3 倍の速度向上が確認できた．特にプログラム 1 の従来手法のみを適用した場合の並列実行結果と提案手法適用時の並列実行結果を比較すると，実行前の並列度の見積りにおいては，提案手法では並列度が 0.5 だけ低下していたが，実際の実行においては提案手法によって 2.7 倍の速度向上が確認できた．プログラム 1，2，3 すべての場合で提案手法適用時の並列実行クロック数が最小となった．これは，本論文で提案したタスク融合によって未適用の場合と比較して，プログラム 1 では同期回数が 17 回から 4 回へ，プログラム 2 では同期回数が 11 回から 2 回へ，プログラム 3 では同期回数が 11 回から 2 回へ削減できたためである．表 5 より 4.4.3 項で述べたコード生成手法がコード生成時間をほとんど増大させることなく，コンパイラ中間表現からデバイスアクセス範囲を抽出する処理を短縮することでコンパイル時間を削減できることが確認できる．特にプログラム 2 において，8.4 倍のコンパイル速度向上が確認できた．

6.4 評価結果のまとめ

本研究で行った実機評価によって，ARM CPU を搭載した実マルチコアボード上で最大 1.4 倍の速度向上が確認できた．特に評価に用いたプログラム 1 は，表 4 における机上計算では本研究で提案した MT 融合手法を用いない場合に最も速度向上が見込めた．しかしながら，実機を用いた評価では並列処理による速度向上が得られなかった．一方で，机上計算においては提案手法を用いない場合よりも並列度が低くなり速度向上が小さくなると見積もられていた提案手法適用版は逐次実行と比較して 1.2 倍の速度向上が確認できた．以上の結果から，机上計算では得られなかった実マルチコアボードの並列化オーバヘッドの影響の大きさとそれを克服するための提案 MT 融合手法の有効性を確認できた．

表 6 3.2.2 項と 5.2 節で示した MT 融合有無による MT 個数の変化
Table 6 Difference in the number of MTs with and without MT fusion methods described in Section 3.2.2 and Section 5.2.

	MT 融合無効化時の MT 個数	MT 融合有効化時の MT 個数
プログラム 1	142	24
プログラム 2	98	10
プログラム 3	98	10

さらに表 5 より，4.4.3 項で述べたコード生成手法適用によってプログラム 1，2，3 に対して 2.2 倍，8.4 倍，6.5 倍のコンパイル速度向上がそれぞれ確認できた．

3 プログラムともラダーステップ数は同等であったが，プログラム 2 と 3 ではプログラム 1 と比較して大きな速度向上を確認できた．これは MT 融合により削減された MT 個数の差によるものである．本研究で提案したコンパイル時間削減手法はコンパイル開始時点での各 MT に配列アクセス範囲情報を付与するため，MT 融合を行わない場合には配列アクセス範囲情報の集約にかかる時間を短縮可能である．一方，コンパイル中に MT 融合が行われた場合，その融合 MT に対して図 9 で示したような集約処理を行わなければならない．**表 6** より，プログラム 1 では MT 融合により 118 個の MT が融合され，プログラム 2，3 では 88 個の MT が融合されている．このため，プログラム 1 のほうがプログラム 2，3 と比較して集約処理の回数が多かったために速度向上率が低くなった．また，融合前後ともにプログラム 1 のほうが MT 数が多いため，本手法で短縮不可能な MT 間の依存チェックの時間が長かったことも速度向上率が低かった原因である．

MT 数が同等であったプログラム 2 と 3 についても速度向上率の差が確認できた．これはプログラム中の配列アクセスパターンの違いが原因である．4.4.3 項で述べたとおり，プログラム中の配列アクセスが不連続である場合，配列アクセス範囲情報の個数が増大する．**表 7** より，プログラム 2 の配列アクセス範囲情報の個数は 3,156 個であり，

表 7 評価プログラムに含まれる配列アクセス範囲情報の個数
Table 7 Number of array range information in the evaluation programs.

	配列アクセス範囲情報の個数
プログラム 1	2,775
プログラム 2	3,156
プログラム 3	2,308

プログラム 3 の配列アクセス範囲情報の個数は 2,308 個であった．以上より，プログラム 2 はプログラム 3 と比較して配列アクセスに連続性がなく配列アクセス範囲情報の個数が多かったために，提案手法適用前では配列アクセス範囲集約に長い時間がかかっていたが，提案手法適用によりその長い集約時間が短縮されたため，より大きな速度向上を得られた．

7. 関連研究

この章では，PLC の高速化，制御プログラムの高速化およびコンパイル時間削減に関連した研究を概説する．

PLC の並列化と高速化に関連した研究としては Vasu らが提案したラダープログラム並列化の研究が存在する [12]．Vasu らの手法ではラダープログラムを独自の中間表現へ変換し，それらを回路ブロック単位で解析する．解析により，他の回路ブロックと依存関係のない並列実行可能な独立した回路ブロックと，他の回路ブロックと依存関係があり並列実行ができない回路ブロックの 2 グループに分けることで並列ラダープログラムを生成している．堀口らの研究では，3 つ以上の接点演算に対して実行前に真理値表を作成しておき，実行時の論理演算を削減する手法が提案された [10]．梶らの研究では，ラダープログラムの論理演算結果が 0 だった場合と 1 だった場合に対応する実行命令列を作成しそれらを条件ジャンプ命令で接続することで実行命令数を削減する手法が提案された [11]．上記 3 つの研究 [10], [11], [12] ではいずれも性能評価において，本論文 6 章で行ったような実機による評価は行っていない．

タスクコストの細かい制御系プログラムの並列化に関連した研究としては，鍾らの MATLAB/Simulink コードに対するクラスタリング手法が存在する [24]．この先行研究では MATLAB/Simulink で定義される制御周期や機能定義を参照し，同一機能定義や同一制御周期を持つブロックを同じコアへ割り当てることでコア間の通信オーバヘッドを削減している．

PLC 向けのプログラミング言語を他の言語へ変換する先行研究としては，インストラクションリスト（IL）を SystemC へ変換し検証する手法 [25]，ストラクチャードテキスト（ST）を ANSI C へ変換し検証する手法 [26]，ラダー図から Arduino C/C++を生成する手法 [27] が存在する．上記のいずれも OSCAR コンパイラで粗粒度並列化を行うための Parallelizable C 規約 [14] は考慮されておらず，実行性能の評価も行われていない．

効率的な配列アクセス範囲解析に関連する研究としては Polyhedral model [28] や LMAD [29] が提案されてきた．しかしながら，これらの研究は内部の配列アクセスパターンがループ誘導変数による線形一次式で表現できるループを対象としており，配列添え字にループ誘導変数を含まないラダープログラム解析には適さない．

分割コンパイルによってコンパイル時間を削減した際のコンパイラの解析性能向上に関連した研究としては，リンク時最適化と呼ばれる手法が提案されてきた [30], [31], [32]．この手法は，分割コンパイル時に各コンパイル単位から生成されるオブジェクトに対して中間表現等の解析情報を付与し，オブジェクトをリンクする際にそれらの解析情報を用いて最適化を行う手法である．これにより分割コンパイルによるコンパイル時間削減とコンパイラによる最適化を両立できる．一方でこれらの手法は，本論文で対象としているコンパイル時の解析時間短縮を目的としていない．

8. まとめ

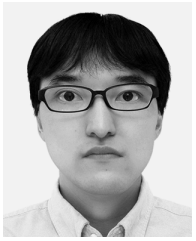
本論文では PLC の高速化に向けたラダープログラムの C 言語変換手法と，タスク融合によるラダープログラムの並列化手法を提案した．ラダープログラムの並列処理は従来実マルチコア上での並列化事例の報告がなかった非常に困難な課題であるが，提案手法の有効性・実用性評価を産業界から提供された実ラダーアプリケーションを用いて，実マルチコアプロセッサ上で行った．評価の結果，1,872 ステップからなる大規模プログラムに対し，ARM プロセッサ 2 コア上で 1.4 倍の速度向上を確認することができた．また 4.4.3 項で示したコンパイル時間削減手法を Intel マシン上で評価を行い，最大で 8.4 倍のコンパイル速度向上を確認した．

以上の評価結果から，提案手法によってラダープログラムと等価な C プログラムが実マルチコアボード上での並列実行時に高速化することが示された．また，ラダープログラムの定数アクセスが頻出する特性に着目することで並列化コンパイラによる並列性抽出が可能であること，およびプログラム解析時間を削減可能なことを確認した．

参考文献

[1] Bayindir, R. and Cetinceviz, Y.: A water pumping control system with a programmable logic controller (PLC) and industrial wireless modules for industrial plants – An experimental setup, *ISA Transactions*, Vol.50, No.2, pp.321–328 (2011).
[2] Saad, N. and Arrofiq, M.: A PLC-based modified-fuzzy controller for PWM-driven induction motor drive with constant V/Hz ratio control, *Robotics and Computer-Integrated Manufacturing*, Vol.28, No.2, pp.95–112 (2012).

[3] Aydogmus, O. and Talu, M.F.: A vision-based measurement installation for programmable logic controllers, *Measurement*, Vol.45, No.5, pp.1098–1104 (2012).
[4] Greeff, G. and Ghoshal, R.: *Practical E-manufacturing and supply chain management*, Elsevier (2004).
[5] 高橋仁之，柴田一樹：C 言語コントローラ/パソコン組み込み型サーボシステムコントローラ，三菱電機技報，Vol.88, No.4, pp.241–244 (2014).
[6] 坂本英幸，水守　隆：半導体製造装置制御システムの構築，横河技報，Vol.47, No.3, pp.91–94 (2003).
[7] 佐藤　隆，吉田栄治，掛林康典ほか：クラスタ構造を持つ半導体製造装置向けの柔軟性の高い制御システム CTCSS（Cluster Tool Controller Software System），電気学会論文誌 D（産業応用部門誌），Vol.124, No.2, pp.160–167 (2004).
[8] 大岩孝彰，勝木雅英：超精密位置決めにおけるアンケート調査—精密メカトロニクスと精密計測に関するアンケート調査，精密工学会誌，Vol.81, No.10, pp.904–910 (2015).
[9] Bolton, W.: *Programmable logic controllers*, Newnes (2015).
[10] 堀口雄揮，梶　夢敏，井口幸洋：PLC の高速化に関する研究 (4)—PLC 用 MPU アーキテクチャと専用コンパイラについて，研究報告システムと LSI の設計技術（SLDM），Vol.2019-SLDM-187, No.46, pp.1–7 (2019).
[11] 梶　夢敏，堀口雄輝，井口幸洋：PLC の高速化に関する研究 (5)—プリコンピューティングによる実行命令数の削減，研究報告システムと LSI の設計技術（SLDM），Vol.2019-SLDM-187, No.47, pp.1–6 (2019).
[12] Vasu, P., Chouhan, H. and Naik, N.: Design and implementation of optimal soft-programmable logic controller on multicore processor, *2017 International Conference on Microelectronic Devices, Circuits and Systems* (*ICMDCS*), pp.1–4, IEEE (2017).
[13] Kasahara, H., Honda, H., Mogi, A., et al.: A multigrain parallelizing compilation scheme for OSCAR (optimally scheduled advanced multiprocessor), *Languages and Compilers for Parallel Computing: 4th International Workshop*, pp.283–297, Springer (1992).
[14] Mase, M., Onozaki, Y., Kimura, K. and Kasahara, H.: Parallelizable C and Its Performance on Low Power High Performance Multicore Processors, *15th Workshop on Compilers for Parallel Computing 2010* (2010).
[15] 津村雄太，川角冬馬，見神広紀ほか：OSCAR 自動並列化コンパイラを用いたラダープログラムの並列性解析，研究報告組込みシステム（EMB），Vol.2022-EMB-59, No.53, pp.1–8 (2022).
[16] Kawasumi, T., Tsumura, Y., Mikami, H., et al.: Parallelizing Factory Automation Ladder Programs by OSCAR Automatic Parallelizing Compiler, *International Workshop on Languages and Compilers for Parallel Computing*, pp.123–138, Springer (2023).
[17] 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 D，Vol.73, No.12, pp.951–960 (1990).
[18] 小幡元樹，白子　準，神長浩気ほか：マルチグレイン並列処理のための階層的並列性制御手法，情報処理学会論文誌，Vol.44, No.4, pp.1044–1055 (2003).
[19] Umeda, D., Kanehagi, Y., Mikami, H., et al.: Automatic parallelization of hand written automotive engine control codes using OSCAR compiler, *17th Workshop on Compilers for Parallel Computing* (*CPC2013*) (2013).
[20] 笠原博徳，合田憲人，吉田明正ほか：Fortran マクロデータフロー処理のマクロタスク生成手法，電子情報通信学会論文誌 D，Vol.75, No.8, pp.511–525 (1992).
[21] Aho, A.V., Lam, M.S., Sethi, R. and Ullman, J.D.: *Compilers: Principles, Techniques, and Tools* (*2nd Edition*), Addison-Wesley Longman Publishing Co., Inc., USA (2006).
[22] Xilinx: Zynq UltraScale+ MPSoC ZCU102 Evaluation Kit, available from ⟨https://www.xilinx.com/products/boards-and-kits/ek-u1-zcu102-g.html⟩ (accessed 2023-05-17).
[23] Kimura, K., Mase, M., Mikami, H., Miyamoto, T., Shirako, J. and Kasahara, H.: OSCAR API for Real-Time Low-Power Multicores and Its Performance on Multicores and SMP Servers, *Languages and Compilers for Parallel Computing*, Gao, G.R., Pollock, L.L., Cavazos, J. and Li, X. (Eds.), pp.188–202, Springer Berlin Heidelberg (2010).
[24] 鍾　兆前，枝廣正人：組込み制御システムに対するマルチコア向けモデルレベル自動並列化手法，情報処理学会論文誌，Vol.59, No.2, pp.735–747 (2018).
[25] Sülflow, A. and Drechsler, R.: VERIFICATION OF PLC PROGRAMS USING FORMAL PROOF TECHNIQUES, *Formal Methods for Automation and Safety in Railway and Automotive Systems*, pp.43–50, Springer (2008).
[26] Sadolewski, J.: Conversion of ST Control Programs to ANSI C for Verification Purposes, *E-informatica: software engineering journal*, Vol.5, No.1, pp.65–76 (2011).
[27] Peterson, D.: Recreating PLC Ladder Logic in an Arduino C/C++ IDE, available from ⟨https://control.com/technical-articles/recreating-plc-ladder-logic-in-an-arduino-c-c-ide/⟩ (accessed 2023-09-02).
[28] Feautrier, P.: Dataflow analysis of array and scalar references, *International Journal of Parallel Programming*, Vol.20, pp.23–53 (1991).
[29] Paek, Y., Hoeflinger, J. and Padua, D.: Efficient and Precise Array Access Analysis, *ACM Trans. Program. Lang. Syst.*, Vol.24, No.1, pp.65–109 (online), DOI: 10.1145/509705.509708 (2002).
[30] Cilio, A.G. and Corporaal, H.: Link-time effective whole-program optimizations, *Future Generation Computer Systems*, Vol.16, No.5, pp.503–511 (online), DOI: 10.1016/S0167-739X(99)00127-2 (2000).
[31] Johnson, T., Amini, M. and David Li, X.: ThinLTO: Scalable and incremental LTO, *2017 IEEE/ACM International Symposium on Code Generation and Optimization* (*CGO*), pp.111–121 (online), DOI: 10.1109/CGO.2017.7863733 (2017).
[32] Kawasumi, T., Tamura, R., Asada, Y., Han, J., Mikami, H., Kimura, K. and Kasahara, H.: Fast and Highly Optimizing Separate Compilation for Automatic Parallelization, *2019 International Conference on High Performance Computing & Simulation* (*HPCS*), pp.478–485 (online), DOI: 10.1109/HPCS48598.2019.9188148 (2019).



川角 冬馬

1995 年生．2018 年早稲田大学基幹理工学部情報理工学科卒業．2019 年同大学院基幹理工学研究科情報理工・情報通信専攻修士課程修了．2019 年同大学院基幹理工学研究科情報理工・情報通信専攻博士後期課程進学，現在に至る．マルチコアプロセッサのアーキテクチャ，コンパイラ，アプリケーションに関する研究に従事．

見神 広紀

1984 年生．2009 年早稲田大学大学院基幹理工学研究科情報理工学専攻修士課程修了，2011 年同大学基幹理工学部助手．現在，同大学グリーン・コンピューティング・システム研究機構研究院客員講師．並列アプリケーション，コンパイラ，マルチコアプロセッサアーキテクチャに関する研究に従事．

吉川 智哉

1990 年生．2014 年神戸大学大学院システム情報学研究科システム科学専攻修士課程修了．同年より三菱電機（株）先端技術総合研究所研究員として勤務．組込み機器の開発や組込みマルチコア並列化技術の研究に従事．

細見 武郎

1982 年生．2007 年東京大学大学院情報理工学系研究科電子情報学専攻修士課程修了．同年三菱電機（株）先端技術総合研究所研究員として勤務．組込みリアルタイムシステム，組込みマルチコア並列化技術の研究に従事．

追立 真吾

1980 年生．2006 年奈良先端科学技術大学院大学情報科学研究科情報システム学専攻修士課程修了．同年より総合電機メーカにて組込みソフトウェア設計開発に従事．2014 年三菱電機（株）入社．組込みリアルタイムシステム，組込みマルチコア並列化技術の研究に従事．

木村 啓二 （正会員）

1972 年生．2001 年早稲田大学大学院理工学研究科電気工学専攻博士課程修了，1999 年同大学理工学部助手，2004 年同大学理工学部コンピュータ・ネットワーク工学科専任講師，2005 年同助教授，2012 年教授，現在に至る．マルチコアプロセッサのアーキテクチャ，コンパイラ，アプリケーションに関する研究に従事．

笠原 博徳 （正会員）

1957 年生．1980 年早稲田大学理工学部電気工学科卒業，1985 年同大学院博士課程修了，工学博士，1985 年カリフォルニア大学バークレー客員研究員，日本学術振興会第一回 PD 特別研究員．1986 年同大学理工学部専任講師，1997 年同教授，現在，情報理工学科教授．2018～2022 年早稲田大学副総長．現在日本工学アカデミー理事・産業競争力懇談会理事・日本学術会議連携会員．情報処理学会 ARC 主査・会誌 HWG 主査・論文誌 HG 主査，IEEE-CS（Computer Society）2018 年会長・2009～2015 年理事，2017～2019 実行役員，IEEE 2018 年技術委員等歴任．情報処理学会 1997 年坂井記念特別研究賞・2020 年功績賞，IEEE Computer Society 2010 年 Golden Core Member Award，2019 年 Spirit of the Computer Society Award，2014 年文部科学大臣表彰科学技術賞（研究部門），1987 年 IFAC World Congress Young Author Award，2020 年 SCAT 会長大賞等受賞．IEEE-CS，ACM，電子情報通信学会等各会員．IEEE Life Fellow．本会フェロー．