Vol.2022-ARC-248 No.53
Vol.2022-SLDM-198 No.53
Vol.2022-EMB-59 No.53
2022/3/11

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

津村 雄太[1,a)]　川角 冬馬[1]　見神 広紀[1]　川上 大樹[2]　細見 武郎[2]　追立 真吾[2]　木村 啓二[1]
笠原 博徳[1]

概要：工場の自動化に伴い，製造装置の制御に使われる PLC(プログラマブル・ロジック・コントローラー)が広く普及している．PLC は従来の電磁リレーと比べて制御内容の変更が容易であり，保守性の面で優れている．PLC の制御プログラムとしてはラダープログラムが主流である．ラダープログラムは論理演算を主体とする条件部と条件によって実行される命令を含む結果部からなる回路ブロックが順に並んで構成されている．製造装置の大規模化・複雑化と共にラダープログラムも大規模化しつつある．その一方で、PLC で使用する CPU は信頼性・耐久性の面から高周波数化による高速化は適せず，マルチコアによる高速化が必要であるが，多くの並列処理が対象とするループ並列性はラダープログラムにはほぼ存在しない．そこで本稿では，ラダープログラムから処理間の並列性を OSCAR 自動並列化コンパイラにより抽出する手法を提案する．本手法では，OSCAR コンパイラによる並列性抽出のため，まずラダープログラムを C プログラムに変換する．その後，コンパイラのプログラムリストラクチャリング機能により条件分岐の隠蔽とタスク粒度の調整を行い，並列性解析を行う．ラダープログラムを C プログラムに自動変換する処理系を開発し，駐車場制御プログラム，ボトルのパッキングプログラム，及び PC 部品の製造プログラムを用いて評価した．評価の結果，これらのプログラムから 3.6 倍、4.0 倍、1.2 倍の並列性をそれぞれ抽出でき，OSCAR 自動並列化コンパイラによるラダープログラムの並列性抽出が有効であることを確認した．

1. はじめに

工場の製造装置の制御手法として PLC が広く普及している．従来のシーケンス制御では，リレーやスイッチなどを使用して，機器間を配線によって組み立てていたが，PLC ではそれらをプログラムによる制御に置き換えることによって，制御内容の変更が容易になり，省スペースで配置でき，配線などの消耗を考慮する必要がなくなった．PLC の制御プログラムとしてはラダープログラムが主流であり，従来のリレー回路を元に作られたプログラミング言語であるため，既存のシーケンス制御と同等に扱うことができる．ラダーは論理演算を主体とする条件部と条件によって実行される命令を含む結果部からなる回路ブロックから構成されている．

PLC を扱うファクトリーオートメーション分野では，PLC の高速化が期待されている．PLC を高速化することによって，1 つの PLC で多数の入力部品や出力部品を制御することが可能になる．また，高速化によるレスポンスの向上によってセンサが入力を確実に検知するために入力を持続する時間を短縮できる．さらに近年では，製造装置の大規模化・複雑化や，製造製品の高性能化に伴いラダープログラムも大規模化しつつあるため高速化要求は高まっている．その一方で、PLC で使用する CPU は信頼性・耐久性の面から高周波数化による高速化は適さず，マルチコアによる高速化が必要であるが，多くの並列化手法で利用されるループ並列性はラダープログラムには乏しい．加えて，マルチコアプロセッサ活用によって，過酷な環境で使用される PLC への発熱の抑制も期待されている．

既存のラダープログラムの高速化手法としてラダープログラムの論理演算に注目し，実行命令数の削減による高速化手法や，マルチコアプロセッサを用いた並列化が挙げられる．しかし，これらの手法ではオフセットアクセスや複数点アクセスなどのラダー命令への高速化が難しく，プログラムリストラクチャリングなどによる高度な高速化手法を適用することができない．

本稿では，ラダープログラムを C 言語によるプログラムに変換し，これを OSCAR 自動並列化コンパイラを用いたマルチコア上での並列実行のための自動プログラムリスト

[1] 早稲田大学
Waseda University.
[2] 三菱電機株式会社
Mitsubishi Electric Corporation.
[a)] tmtmwaseda@kasahara.cs.waseda.ac.jp

Vol.2022-ARC-248 No.53
Vol.2022-SLDM-198 No.53
Vol.2022-EMB-59 No.53
2022/3/11

ラクチャリング及び並列性解析を行う手法を提案する．具体的には，ラダープログラムから C プログラムを自動生成する処理系を開発し，生成されたプログラムに対して，OSCAR 自動並列化コンパイラが有する条件分岐の隠蔽や粒度の調整などのプログラムリストラクチャリング機能を適用し，ポインタ解析などの高度な並列性解析により，処理間の並列性の抽出を行い，一つ一つの処理が小さいという制御系プログラムの特徴に対して有用な並列化手法を実現した．

以下本稿では，2 節で関連研究を概観する．3 節でラダープログラムについて説明し，4 節で OSCAR 自動並列化コンパイラによる並列性抽出手法，5 節でラダープログラムから C 言語への変換手法をそれぞれ説明する．6 節で性能評価について報告し，7 節でまとめる．

2. 関連研究

PLC の高速化要求に対して，自動で高速化を行う手法がいくつか提案されている．ラダープログラムは論理演算を主体とする条件部と条件によって実行される命令を含む結果部から構成されており，論理演算部分に注目し演算回数を削減する手法が提案されている．

堀口等の PLC 高速化手法 [1] では 3 つの論理演算に対応する真理値表を予め用意することで複数の接点命令の処理にかかるステップを軽減することができる．さらに，専用のアーキテクチャを構成し，2 つの MPU を接続することで並列実行による高速化も実現した．

梶等の PLC 高速化手法 [2] ではラダープログラムが 0 か 1 のみを扱う論理デバイスを数多く持つ性質を利用し，ある論理デバイスが 0 または 1 を代入した場合の命令列を生成し，それをジャンプ命令で結合する．生成された命令列では，論理デバイスの値が決まっているため，そのデバイスに関わる命令が省略できるため，実行命令数が省略されたソースコードを生成できる．

マルチコアによる並列化手法も提案されている．Vasu 等の PLC 高速化手法 [3] ではラダープログラムを中間表現に変換し，Python プログラムを用いてラダープログラムを回路ブロックごとに依存性解析を行い，Python のマルチプロセスによる実行モデルを利用することで並列化による高速化を実現した．

制御プログラムの並列化に関しては, MATLAB/Simulink 等のモデルベース設計に対する研究がある．鍾等 [4] は，モデル内のブロック間の並列性を用いることで並列化を実現している．梅田等 [5] は，モデルのブロック間の並列性に加えて，ブロック内部の並列性を抽出することで，高速化を実現している．ラダープログラムはモデルベース開発と同様に，制御内容を図として表現するが，制御の流れしか記載されていないため，プログラム構造の把握とそれによる並列性の抽出が困難である．この問題に対して，本稿ではラダープログラムを C 言語変換し，OSCAR 自動並列化コンパイラで解析することで，高度な並列性解析とプログラムのリストラクチャリングを流用することで，任意のコア数向けに最適化された並列 C コードを自動生成することができる．また，コンパイラによってプログラムのデータ依存を示すマクロタスクグラフを生成することでプログラム構造を容易に確認可能にした．

3. ラダープログラム

本節ではラダープログラムの概要と構成要素やラダー命令，ラダー図の実行順序について述べる．

3.1 ラダープログラムの概要

ラダープログラムとは，従来のリレーやスイッチなどを使用して行っていたシーケンス制御をモデル化したもので，制御回路をそのまま図として表現したラダー図と，図表現をテキストとして表現したインストラクションリスト (IL) がある．開発者は一般に開発ツール上でラダー図としてプログラム開発を行い，ツールによって IL の出力も可能である．ラダー図，IL をそれぞれ**図 1**，**図 2** に示す．

3.2 ラダープログラムの構成要素

ラダー図は両端にある母線をラダー命令で繋いで表現する．左側の母線に接続された命令から始まり，右側の母線に接続された命令で終わる回路を回路ブロックと呼ぶ．ラダー図はこれらの回路ブロックを繋いで構成される．回路ブロック内では，ある条件が成立した場合にある命令が実行されるという制御を表現している．ラダー命令ではデバイスと呼ばれるメモリの値を参照する．代表的なデバイスとして，論理値を持つ入力 X や出力 Y，PLC 内部でビット情報を保持する内部リレー M や，16 ビットのワードデータを扱うワードデバイス D，時間を計測するタイマデバイス T や，数値をカウントするカウンタデバイス C などがある．デバイスはデバイス記号と数字の組により表現され，

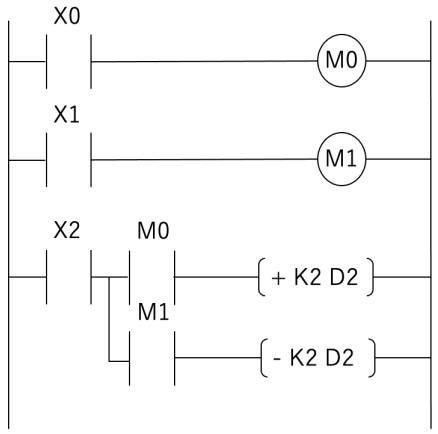

図 1 ラダー図の例

Vol.2022-ARC-248 No.53
Vol.2022-SLDM-198 No.53
Vol.2022-EMB-59 No.53
2022/3/11

これによりデバイスのどの場所の値にアクセスするかを表現する．X0 の場合は入力デバイス X の 0 番目の要素の値を利用する．

3.3 ラダーで使用される命令

回路ブロックの条件部では**図 3** の (1)“a 接点” や (2)“b 接点” が使われる．a 接点は X0 が 1 の時に両端が導通し，b 接点は X0 が 0 の時に両端が導通する．回路ブロックの実行部では (3)“OUT 命令” や (4)“実行命令” が使われる．OUT 命令は常に実行される命令であり，条件が成立したら M0 に 1 を保持し，成立しなかった場合は 0 を保持する．実行命令は条件が成立した時に実行される命令であり，データ転送命令や四則演算など様々な命令が実行される．

ラダー図で使用される記号は IL 内では，LD や AND,OUT,+などの IL 言語で表現される．条件部では接点開始命令として a 接点，b 接点それぞれを扱う LD,LDI があり，接点同士を結合する AND,OR,ANI,ORI がある．実行部ではそれぞれの命令を表す IL 言語が使用され，(3) の条件部によって 0 または 1 を保持する OUT 命令や加算の+命令や減算の-命令，データ転送を行う MOV 命令などがある．

3.4 ラダープログラムの実行順序

ラダープログラムは左から右に，上から下に実行される．そして END 命令を実行するとまた一番上から下へと実行される．先頭の命令から END 命令までの実行時間をスキャンタイムといい，このスキャンタイムの高速化が求められている．

実際に図 1 を例として説明する．ラダー図の 1 行目では，X0 の値を読み込み，その結果を M0 に格納する．2 行目では，X1 の値を読み込み，その結果を M1 に格納する．3 行目では，まず，X2 の値を読み込み，M0 の値と AND を取り，その結果が 1 であれば [+ K2 D2] を実行し，0 であれば実行しない．同様に，4 行目では X2 の値と M1 の値と AND を取り，その結果が 1 であれば [- K2 D2] を実行する．つまり，X2 が 1 の場合，M0 が 1 になると D2 の値が 2 増え，M1 が 1 になると D2 の値が 2 減るプログラムである．

ステップ番号	命令	デバイス
0	LD	X0
1	OUT	M0
2	LD	X1
3	OUT	M1
4	LD	X2
5	MPS	
6	AND	M0
7	+	K2
		D2
10	MPP	
11	AND	M1
12	-	K2
		D2
15	END	

図 2 図 1 をから生成したインストラクションリスト

4. OSCAR 自動並列化コンパイラ

本節では OSCAR 自動並列化コンパイラ [6], [7] で用いた，粗粒度タスク間の並列処理と，タスク間のデータ依存性を表すマクロタスクグラフについて述べる．加えて，条件分岐の隠蔽や粒度の調整などのプログラムリストラクチャリング機能について述べる．

4.1 粗粒度タスク並列処理とマクロタスクグラフ

OSCAR 自動並列化コンパイラでは，従来のループ並列化手法 [8],[9] に加えて，プログラム全域からループやサブルーチン間の並列性を抽出する粗粒度タスク並列処理 [10], [11] が可能である．

粗粒度タスク並列処理では，ラダープログラムから生成された C プログラムを基本ブロック (bb)，繰り返しのあるブロック (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 は代入文や四則演算，条件分岐などが連続するブロックであり，rb は for ループなどを含むブロックであり，sb は関数呼び出しを含むブロックである．プログラムのマクロタスクへの分割後，マクロフローグラフ (MFG) を生成する．MFG は，プログラムの解析によって得られた，条件分岐などによるプログラムの実行の流れを表すコントロールフローと，MT 間の変数や配列の依存関係を表すデータ依存を表す．そして，MFG 内のコントロールフローとデータ依存から，各 MT が最も早く実行可能になる最早実行可能条件解析を行い，各 MT の並列性を表現したマクロタスクグラフ (MTG) を生成する．

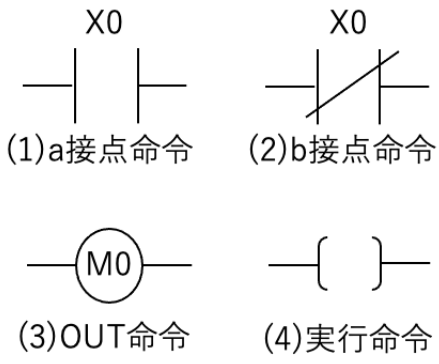

図 3 ラダー図で使用する記号

Vol.2022-ARC-248 No.53
Vol.2022-SLDM-198 No.53
Vol.2022-EMB-59 No.53
2022/3/11

図 1 から生成された C プログラムである**図 5** から生成されたマクロタスクグラフを**図 4** に示す．図中の各ノードはマクロタスクを表しており，bb1 や bb3 などのノード内の小円は条件分岐を表している．ノードから出ている各エッジは MT 間の関係を表しており，実線のエッジはデータ依存を表し，点線のエッジはコントロール依存を表す．また，これらのエッジを束ねるアークはエッジ同士の関係を表し，実線は AND の関係，点線は OR の関係を表す [12]．例えば，bb6 と bb7 が 12 行目と 13 行目のプログラムに対応しており，bb6 の条件によって bb7 が実行されるため点線エッジで繋がれている．また，bb6 は bb3 と M[0] でデータ依存が存在するため実線エッジで繋がれている．

マクロタスクグラフ生成後，各 MT はマルチコア内の各コアに割り当てられ並列実行される．この割り当ての際，コアへのタスクスケジューリング方法は，ダイナミックスケジューリングとスタティックスケジューリングから選択される．ダイナミックスケジューリングはプログラム実行時に割り当てを決定し，スタティックスケジューリングはプログラムの解析時に割り当てを決定する．ダイナミックスケジューリングの場合，条件分岐による実行時の変化に対応して MT を割り当てることができる．一方で，スタティックスケジューリングは条件分岐がない場合にあらかじめスケジューリングできるため実行時のオーバーヘッドがない．

4.2 制御プログラムの粗粒度タスク並列処理のための最適化手法

本項では，粗粒度タスク並列処理の性能を向上させるための最適化手法のうち，本稿のラダープログラムの高速化に適用した基本ブロック分割，マクロタスク融合，条件分岐複製について述べる．

4.2.1 基本ブロック分割

4.1 節で述べたように，基本ブロック bb は代入文や四則演算，条件分岐などが連続するブロックである．bb 内のステートメント間に依存がない場合，それらを別々の基本ブロックへ分割配置することで，より多くの並列性を抽出することができる．

4.2.2 マクロタスク融合

ラダーのような制御プログラムの場合，センサ等の条件によって動作が決まる処理が多く，一つ一つの MT のコストが小さいものが多い．そのため，ダイナミックスケジューリングを用いたタスクの割り当てを行った場合，スケジューリングコストが相対的に大きくなり，並列処理効率が低下してしまう．そこで，粗粒度タスクを融合し条件分岐を隠蔽することで，実行コストを大きくしつつスケジューリングコストを無くせる手法が有効である．具体的には，条件分岐と分岐先のタスクを一つのマクロタスクに融合することで，タスク内に条件分岐を隠蔽することができる．そのため，マクロタスクグラフから条件分岐を取り除くことができ，スタティックスケジューリングが可能になる [13]．図 4 に基本ブロック分割及びマクロタスク融合を適用させたマクロタスクグラフを**図 6** に示す．図 4 の bb1 から bb3 が表す 1 から 7 行目のプログラムのうち，1 から 4 行目が図 6 の bb1,block2,bb3 になり，bb5 から始まる一連の条件分岐が block7 へと融合された．

4.2.3 条件分岐複製

マクロタスク融合で，条件分岐を隠蔽することによって，スケジューリングコストを削減することができたが，これによって，条件分岐内の並列性が失われてしまう可能性がある．そこで，条件分岐内の並列性を抽出するための手法が提案されている [14]．具体的には，あらかじめ条件分岐タスクの条件式を複製し，分岐内のタスクを MT ごとに条件分岐ブロックを生成することで条件分岐内の MT を並列実行する．

5. ラダープログラムから C 言語への変換

本節では，ラダープログラムから C プログラムを生成する変換処理系の概要について述べる．

5.1 ラダープログラムと C 言語の対応関係

本項では，変換処理系におけるラダープログラムと C 言

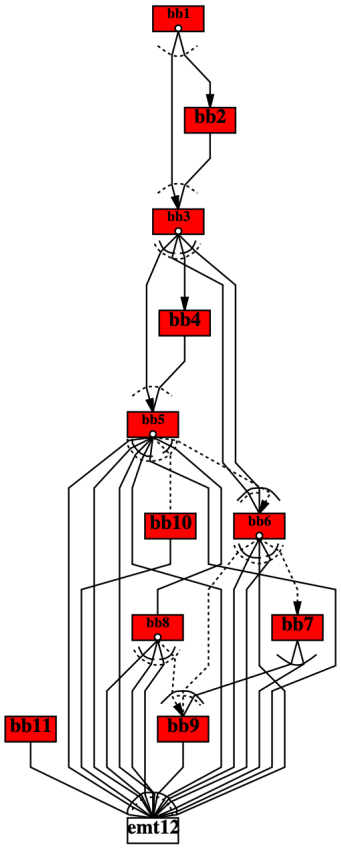

図 4 OSCAR 自動並列化コンパイラによって生成された図 5 のマクロタスクグラフ

語の具体的な対応関係について述べる．ラダープログラムが条件部と実行部に分かれていることに着目し，一つの回路ブロックを 1 つの if 節へと変換することでラダープログラムの C 言語化を実現した．

5.1.1 接点に関連した命令の扱い方

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、スタックロードを行う MRD 命令、スタックポップを行う MPP 命令が存在する．これらの命令を用いることで，例えば図 1 のようなラダープログラムを記述する際に接点 X2 を 2 つの接点に接続可能となり，コード量を削減できる．C コード生成においては，これら 3 つの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コード量削減を行う．具体的には，本研究で作成した変換処理系はこれらの命令を以下のように扱う．

- MPS：直前までの接点演算結果を用いて if 節を開始し，以降の処理をこの if 節の中で実行するようにする
- MRD：直前までの接点演算結果を用いて if 節を出力
- MPP：直前までの接点演算結果を用いて if 節を出力する．直後の実行部の出力終了後，if 節を 1 つ閉じる

具体例として以上のようなルールに則って図 2 のラダー図を図 5 の C コードへ変換する場合について考える．まず，ステップ 5 の MPS 命令により直前の LD 命令を用いた if 節を生成する．続くステップ 6 の AND 命令は MPS 命令によって生成した if 節内部へさらに if 節を追加する命令として解釈する．さらに，ステップ 7 の実行命令の出力後，ステップ 10,11 の MPP,AND 命令によってステップ 5 の MPS 命令が生成した if 節内へもう一つ if 節を追加する．最後にステップ 12 の減算命令出力後，ステップ 11 の AND 命令による if 節とステップ 5 の MPS 命令による if 節の両方を閉じることで，図 5 を得る．このように上述のルールに則ったコード生成を行うことで，冗長な X[0x2] が if 文の条件式に存在しない，元のラダー図との対応関係が容易に取れる C コードが生成可能となっている．上述のルールのようにネスト構造を生成しない場合，図 1 の加算命令と減算命令に対してそれぞれ X[0x2] の冗長なチェックが出力される．

5.1.2 デバイスの表現方法

ラダー内ではデータを扱うためのデバイスを C 言語では配列として表現し，デバイスの番号を配列の要素数とすることで，各デバイスごとに配列の要素数によって依存性の解析を可能にした．また，この表現方法によって，後述する BMOV 命令などの複数点アクセス命令や，オフセットアクセスなどの並列性解析を可能にした．

5.1.3 回路ブロックと if 節の対応関係

本節では，ラダー図から C 言語に変換されるフローと，OSCAR 自動並列化コンパイラによって生成されたマクロタスクグラフの例を示す．

図 1 を対象ラダー図とした場合，まず開発ツールによって生成された IL(図 2) を変換処理系へ入力する．この時，図 2 の LD や AND,MPS,MPP 命令は回路ブロックの実行に関わる部分なので条件部とし，+,-は条件部によって実行される命令として実行部に分類する．そして条件部を if 文の条件式に，実行部を if 文の実行部分にそれぞれ対応させ C プログラムへと変換する．OUT 命令は常に実行される命令なため，そこで保持する値の制御は，一時変数の導入とそこに値を代入する if 文により行う．変換処理系によって生成された C プログラムを図 5 に示す．変換された C プログラム中の各コメントの数字はラダー命令のステップ番号を表す．

コンパイラによって生成された最適化済みのマクロタスクグラフを図 6 に示す．それぞれグラフ内の bb1,block2,bb3 が 1 つ目の回路ブロックに対応し，bb4,block5,bb6 が 2 つ目の回路ブロック，block7 が 3 つ目の回路ブロックに対応している．変換処理系では，OUT 命令は一時変数を 0 初期化する文と条件が真だった場合にその一時変数に 1 を代入する文，if 節終了後に OUT 命令で使われたデバイスに対して一時変数を代入する文という構成で C プログラムへ変換される．ノード間を繋ぐエッジはデータ依存を表しており，今回のプログラムでは，bb3 と block7 が M[0] でデータ依存があり，bb6 と block7 では M[1] でデータ依存が存在する．

5.1.4 実行命令の表現方法

変換処理系では，ラダー命令を C 言語で定義した．その内のいくつかを例として以下に示す．

図 7 にデータ転送命令 MOV を示す．MOV 命令は第一引数の値を第二引数にコピーする命令であるため，C 言語化の場合は代入命令になる．

図 8 に算術演算命令 D+を示す．D+命令は 32bit データ同士の足し算命令で，第一引数と第二引数の演算結果を第二引数に代入する．なお，引数が 16bit データだった場

```
1   Mbrls0sbrr_f0 = 0;
2   if(X[ 0x0 ]){//0
3       MOV(1,Mbrls0sbrr_f0);
4   }
5   M[ 0 ] = Mbrls0sbrr_f0;
6   Mbrls1sbrr_f0 = 0;
7   if(X[ 0x1 ]){//2
8       MOV(1,Mbrls1sbrr_f0);
9   }
10  M[ 1 ] = Mbrls1sbrr_f0;
11  if(X[ 0x2 ]){//4
12      if(M[ 0 ]){//4
13          _PLUS_( 2, D[ 2 ]);//7
14      }
15      if(M[ 1 ]){//10
16          _MINUS_( 4, D[ 2 ]);//12
17      }
18  }
```

図 5 変換処理系による図 2 の C 言語への変換結果

Vol.2022-ARC-248 No.53
Vol.2022-SLDM-198 No.53
Vol.2022-EMB-59 No.53
2022/3/11

合は 2 つの 16bit データを使用して 32bit データを作成したのち演算を行う．また，タイマデバイスの場合は，タイマの値を保持するメンバと，条件部でも使われる接点の情報を保持するメンバからなる構造体配列として表現している．

図 9 にデータ転送命令 BMOV を示す．BMOV 命令は第二引数に第一引数の値を第三引数分コピーする命令であるため，C 言語では通常 memcpy 命令と同等の命令として処理される．しかし，BMOV 命令の第二引数には，図 9 の K4L16017 のような 1bit データを任意の点数分繋げて一つの 16bit データとして扱う桁指定デバイスという表現方法がある．K4L16017 の場合，L16017 を最下位ビットとし，L16032 を最上位ビットとする 16bit のデータを作成し，そのデータに対して R30100 を代入するといった命令を 48 点分行う．

6. 性能評価

本節ではラダープログラムに対する評価結果を述べる．なお，評価方法については，OSCAR 自動並列化コンパイラの解析によって推定されたプログラムの実行時間を用いて推定並列度を算出した．並列度とは，マクロタスクグラフ全体の推定実行時間を，マクロタスクグラフの入口ノードから出口ノードまでの最長のパス長を表すクリティカスパス長で割ったものである [15]．想定する実行環境は Raspberry Pi4 で使用されている Cortex-A72 である．

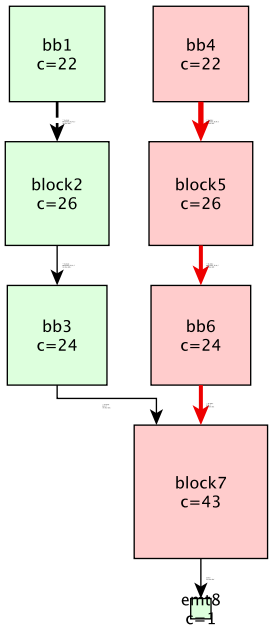

図 6 OSCAR 自動並列化コンパイラによって生成された図 5 の最適化済みマクロタスクグラフ

```
1    MOV M0 M10
2    -> M[10] = M[0]
```

図 7 データ転送命令 MOV の C 言語定義

6.1 評価対象アプリケーション

本評価では，参考書 [16] に掲載されている駐車場制御プログラム及びボトルのパッキングプログラムの 2 つと，三菱電機より提供された PC 部品の製造プログラムの，3 つのラダープログラムを用いた．駐車場制御プログラムは駐車場内の車の位置と集金機械の入金状態に応じて駐車場出入口のバリケードを動かすプログラムである．ボトルのパッキングプログラムはベルトコンベアと自動梱包機が直列につながっている状況を想定し，自動梱包機に入っているボトル本数を監視しながらベルトコンベアと梱包機の制御を行ってボトルを 4 本 1 組にまとめるためのプログラムである．**表 1** に各アプリケーションの概要を示す．表の各値は，各アプリケーションが 1 回実行されるときのステップ数と OSCAR 自動並列化コンパイラによって推定された実行時間である．

```
1    D+ T2 D2
2    ->
3    D_PLUS_timer_cpp(&T[2], &D[2])
4    void D_PLUS_timer_cpp
5         (struct timer_dev *s1, int *d){
6        int S1, D;
7        S1 = ((s1[1].time & 65535)<<16) |
8              (s1[0].time & 65535);
9        D = ((d[1] & 65535)<<16) |
10             (d[0] & 65535);
11       D = D + S1;
12       d[0] = D &65535;
13       d[1] = (D >> 16) & 65535;
14   }
```

図 8 タイマデバイスを使用した算術演算命令 D+の C 言語定義

```
1    BMOV R30100 K4L16017 K48
2    ->
3    void BMOV_digit_assign_1_cpp(int digit,
         int *s, int *d, int d_start, int
         d_offset, int n){
4    int i, j, S, start;
5    start = d_start * 16 + d_offset;
6
7    for(i=0;i<n;i++){
8        S = s[i];
9            for(j=0;j<16;j++){
10               d[start+i*16+j] = (1 & S);
11               S = S >> 1;
12           }
13   }
14 }
```

図 9 桁指定デバイスを含む BMOV 命令の C 言語定義

Vol.2022-ARC-248 No.53
Vol.2022-SLDM-198 No.53
Vol.2022-EMB-59 No.53
2022/3/11

表 1　評価アプリケーションの概要

アプリケーション名	総ステップ数	推定実行時間
駐車場制御	29	0.45ms
ボトルのパッキング	27	0.40ms
PC 部品の製造	7934	747.97ms

6.2　評価結果

表 2 に各アプリケーションの OSCAR 自動並列化コンパイラによる解析結果を示す．表の各値は OSCAR コンパイラが算出した 1 コアでの推定実行時間を 1 とした場合の推定並列度である．解析は以下の 2 条件で行った．

- 最適化 a：4.1 節で述べた粗粒度並列化と 4.2.2 節で述べたマクロタスク融合を適用した場合の解析結果
- 最適化 b：最適化 a に加えて，4.2 節で示した基本ブロック分割及び条件分岐複製を適用した場合の解析結果

表 2　OSCAR 自動並列化コンパイラによる並列性解析結果

アプリケーション	最適化 a	最適化 b
駐車場制御	1.7	3.6
ボトルのパッキング	1.9	4.0
PC 部品の製造	1.1	1.2

以下に，各プログラムの並列性抽出の結果を説明する．

駐車場制御プログラムの解析結果を図 12 に示す．図 10 に駐車場制御プログラムのラダー図を示す．回路ブロック 1 番とマクロタスク 1,2,3 が集金機械の制御，回路ブロック 6 番とマクロタスク 15,16,17 がタイマー制御，これらを除いた部分がバリケード制御にそれぞれ対応している．これら 3 つを粗粒度タスクとみなし，粗粒度並列化を適用することにより 1.7 倍の速度向上が見込めることが確認できた．さらに，基本ブロック分割によって bb11 をデータ依存の無いブロック 2 つに分割することで 3.6 倍の並列性を抽出した．

ボトルのパッキングプログラムの解析結果を図 13 に示す．図 11 にパッキングプログラムのラダー図を示す．回路ブロック 1 番から 4 番までとマクロタスク 1 から 12 がベルトコンベアの制御部であり，回路ブロック 5 番以降とマクロタスク 13 以降が自動梱包機の制御部である．これら 2 つの制御をそれぞれ粗粒度タスクとみなし，粗粒度並列化を適用することにより 1.9 倍の速度向上が見込めることが確認できた．さらに，bb6 と bb9 を基本ブロック分割によってデータ依存の無いブロックに分割することで 4.0 倍の並列性を抽出した．

PC 部品の製造プログラムの解析結果について述べる．**図 14** のような内部のマクロタスク間に依存の無い if 節に対して 4.2.3 節の条件分岐複製を適用させることで，依存の無い if 節同士での並列化が可能になるので，より高い並列性の抽出が可能になる．OSCAR コンパイラによる並列性解析の結果、本プログラムには連続した同一処理など，並列性が乏しい箇所とその並列性抽出を阻害する要因を特定可能なことが確認できた．

7.　まとめ

本稿では，OSCAR 自動並列化コンパイラを用いて，C 言語化されたラダープログラムから並列性の抽出を行い，ラダープログラムの並列性解析を行った．この結果，駐車場制御プログラムでは 3.6，ボトルのパッキングプログラムでは 4.0，PC 部品の製造プログラムでは 1.2 の推定並列度がそれぞれ得られた

以上から，提案手法はラダー図で書かれたプログラムから変換処理系を用いて C プログラムを自動生成し，OSCAR 自動並列化コンパイラで解析することで，ラダープログラム内の並列性を抽出し，自動で並列化可能なことが確認できた．また，OSCAR 自動並列化コンパイラによって，マクロタスクグラフを生成することで，ラダープログラムの構造を視認でき，並列性阻害要因も確認できるため，より

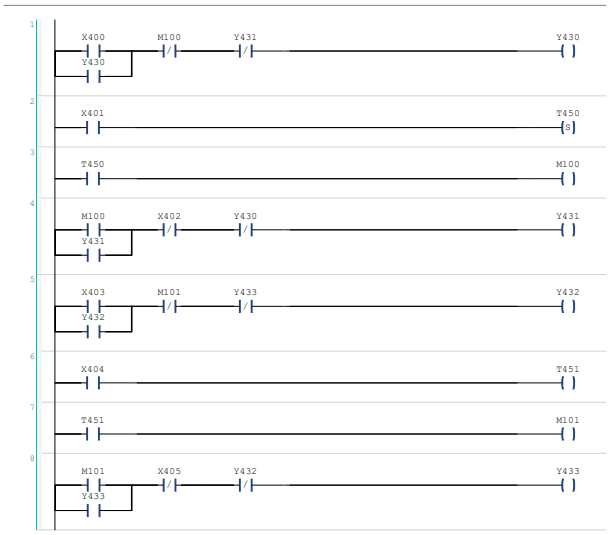

図 10　駐車場制御プログラムのラダー図

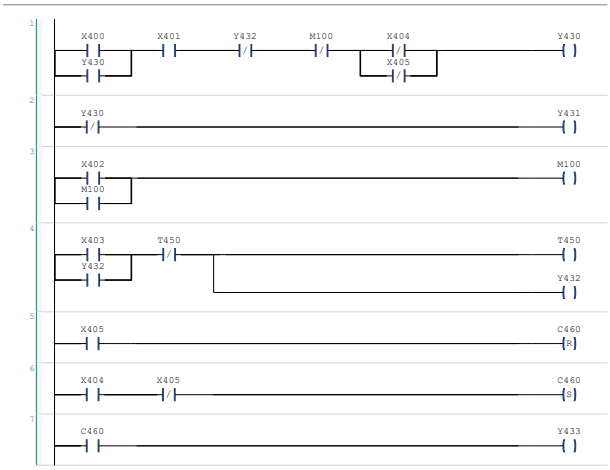

図 11　ボトルのパッキングプログラムのラダー図

ラダープログラムの並列性を生み出すためにプログラマに対しての支援も可能となった.

参考文献

[1] 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-. 電子情報通信学会　信学技報, 2019.
[2] 梶夢敏, 堀口雄揮, 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-. 電子情報通信学会　信学技報, 2019.
[3] P Vasu, Harish Chouhan, and Nitin Naik. Design and implementation of optimal soft-programmable logic controller on multicore processor. In *2017 International conference on Microelectronic Devices, Circuits and Systems (ICMDCS)*, pp. 1–4. IEEE, 2017.
[4] 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, feb 2018.
[5] 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. 718–729, feb 2016.
[6] 笠原博徳, 小幡元樹, 石坂一久. 共有メモリマルチプロセッサシステム上での粗粒度タスク並列処理. 情報処理学会論文誌, Vol. 42, No. 4, pp. 910–920, apr 2001.
[7] オスカーテクノロジー株式会社　ホームページ. `https://www.oscartech.jp`.
[8] Using the GNU Compiler Collection (GCC). `https://gcc.gnu.org/onlinedocs/gcc-11.2.0/gcc/`.
[9] ”Intel® Fortran Compiler. `https://www.intel.com/content/www/us/en/developer/tools/oneapi/fortran-compiler.html`.
[10] 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. 73, No. 12, pp. p951–960, 1990.
[11] 笠原 博徳他. 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, Vol. 75, No. 8, pp. p511–525, 1992.
[12] 笠原博徳. 並列処理技術 / 笠原博徳著. コロナ社, 東京, 1991.6.
[13] 梅田弾, 金羽木洋平, 見神広紀, 林明宏, 谷充弘, 森裕司, 木村啓二, 笠原博徳. MATLAB/Simulink で設計されたエンジン制御 C コードのマルチコア用自動並列化. 情報処理学会論文誌, Vol. 55, No. 8, pp. 1817–1829, aug 2014.
[14] Dan Umeda, Yohei Kanehagi, Hiroki Mikami, Akihiro Hayashi, Keiji Kimura, and Hironori Kasahara. Automatic parallelization of hand written automotive engine control codes using oscar compiler. In *17th Workshop on Compilers for Parallel Computing (CPC2013)*, Jul. 2013.
[15] 準白子, 耕平長澤, 一久石坂, 元樹小幡, 博徳笠原. マルチグレイン並列性向上のための選択的インライン展開手法. 情報処理学会論文誌, Vol. 45, No. 5, pp. 1345–1356, may 2004.
[16] W.Bolton. *Programable Logic Controllers Sixth Edition*. Newnes, 2015.

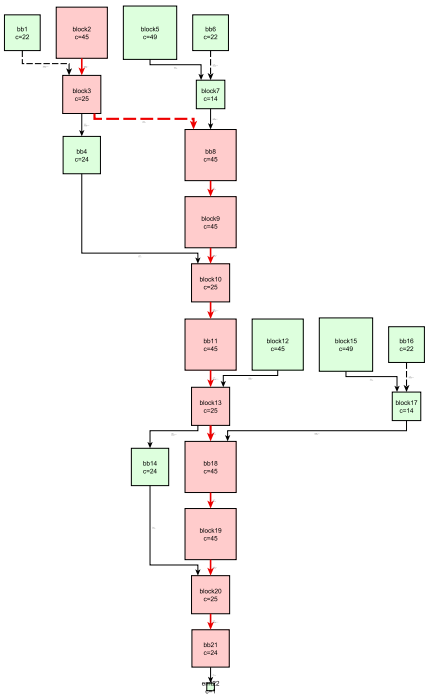

図 12　駐車場制御プログラムの解析結果

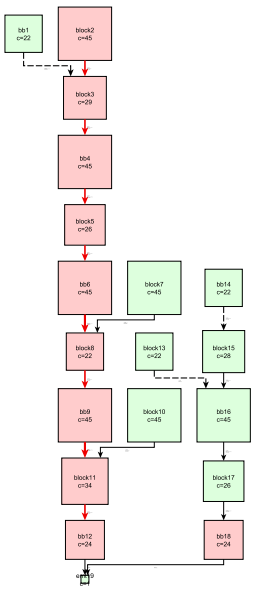

図 13　ボトルのパッキングプログラムの解析結果

```
1  if(SM[ 400 ]){
2      BMOV_digit_assign_1_cpp(16, &R[ 30100 ],
           &L[ 0 ], 1001, 1, 48);
3      BMOV_digit_assign_1_cpp(16, &R[ 30200 ],
           &L[ 0 ], 1051, 1, 48);
4      BMOV_digit_assign_1_cpp(16, &R[ 30300 ],
           &L[ 0 ], 1101, 1, 48);
5      BMOV_digit_assign_1_cpp(16, &R[ 30400 ],
           &L[ 0 ], 1151, 1, 48);
6      BMOV_digit_assign_1_cpp(16, &R[ 30500 ],
           &L[ 0 ], 1201, 1, 48);
7      BMOV_digit_assign_2_cpp(16, &L[ 0 ],
           626, 1, &D[ 0 ], 251, 1, 16);
8      BMOV_digit_assign_2_cpp(16, &L[ 0 ],
           642, 1, &D[ 0 ], 252, 5, 16);
9      BMOV_digit_assign_2_cpp(16, &L[ 0 ],
           658, 1, &D[ 0 ], 253, 9, 16);
10     BMOV_digit_assign_2_cpp(16, &L[ 0 ],
           626, 1, &R[ 0 ], 1875, 0, 48);
11 }
```

図 14　PC 部品の製造プログラム内の条件分岐複製適用 if 節