3A-02

モデルベース開発における KALRAY MPPA メニーコア向け並列化

鍾　兆前†　　枝廣 正人†
†名古屋大学大学院情報科学研究科情報システム学専攻

1　はじめに

近年，車載制御システムの大規模化・複雑化に伴い，MATLAB Simulink などのプラットフォームを利用するモデルベース開発が普及しつつある．一方，プロセッサのマルチコア化，メニーコア化が期待されていて，最近製品化されたメニーコアの中に，KALRAY MPPA というメニーコア [1] がある．MPPA は単一操作当たりのエネルギー消費が低い，時間予測が可能という特徴を持ち，バッテリーで動作し，時間制約が厳しい自動車向けには適していると注目されている．なお，実際にメニーコアの性能を享受するためには，メニーコアの構造を考慮し，制御ソフトウェアを並列実行させることも重要である．そこで，本研究においては，MPPA メニーコアを車載制御に用いるため，ブロック線図で構成される Simulink モデルから MPPA 向けの SigmaC ソースコード生成フローを構築した．さらに，並列化アルゴリズムの研究を進め，Kernighan-Lin 法 [2] とよばれる既存グラフ分割アルゴリズムを実装し，実際の車載モータアプリケーションを用いて評価を行った．

2　用語解説

2.1　MATLAB Simulink モデル

MATLAB Simulink は MathWorks 社が開発した，システム レベル設計とシミュレーション，自動コード生成などの作業をサポートするプラットフォームである．Simulink モデルはブロックで構成され，ブロック間は信号線で接続するブロック線図で記述する．個々のブロックは Simulink モデルの基本構成要素で，簡単な四則計算から，フィードバックやループの制御といった複雑な機能を持つことができる．なお，RTW コーダーは，Simulink のオプション製品で，モデルから逐次実行の C コードと C++コードを生成するツールである．

2.2　KALRAY MPPA-256 メニーコア

MPPA とは，フランスの KALRAY 社が開発したシングルチップメニーコアプロセッサファミリーである．その内，MPPA-256 はチップ上に 16 個の計算クラスタと４個の IO クラスタを統合したプロセッサである．計算クラスタは MPPA アーキテクチャの基本処理ユニットであり，１つの計算クラスタにつき，16 個の計算コアと，１つのシステムコアが含まれる．システムコアはタスクとデータ転送のスケジューリング及び実行を管理する．クラスタ内においてコア間のデータ交換は共有メモリで行い，クラスタ内の通信時間は短い．クラスタ間のデータ転送は NoC を介し行うため，クラスタ間の NoC 通信時間はかなり長い．MPPA の動作周波数は 400 MHz であり，1 サイクルは 2.5ns である．

2.3　SigmaC データフロープログラミング言語

SigmaC とは，KALRAY 社が開発した MPPA メニーコアにおけるシクロ静的データフロー計算モデルベースの拡張言語であり [3]，SigmaC アプリケーションは，タスクで構成されるデータフローグラフの形に表現される．SigmaC タスクはクラスタレベルの割り当てが可能であり，開発者がタスクを MPPA の計算クラスタ・I/O クラスタに配置させることにより並列実行することが可能である．しかし，クラスタ内部のコアレベルの割り当てはシステムコアが管理するため，タスクの実行コアの指定ができない．なお，クラスタ割り当て情報はソースコードに書くのではなく，手動で割当てするか，マッピングファイルを介することで，クラスタ割り当てを行う．

3　考案した並列化フロー

Simulink モデルを実際のメニーコアに実装するには，モデルからの情報抽出，ソースコードの生成とタスクの並列実行などの課題がある．それに対して， MPPA の SigmaC 言語は，同期処理が容易で，Simulink モデルのブロック線図と類似なデータフローグラフで構成しているという特徴があり，これらの特徴を活かして，ブロックレベルの並列化が可能である．本研究において，Simulink モデルを KALRAY MPPA-256 メニーコアに実装するため，いままで開発されたツールを統合し，MPPA-256 向け SigmaC コード生成フローを構築した．

考案したフローにおいて，まず，MATLAB の環境で，入力された Simulink モデルからブロック構造情報を抽出して，各ブロックの構造情報と，RTW コーダー

Parallelization for KALRAY MPPA MANYCORE in model-based development
†Zhaoqian ZHONG　†Masato Edahiro
†Department of Information Engineering, the Graduate School of Information Science, Nagoya University

より生成した逐次処理 C コードにある相応の実行コード断片を探しだし，コード情報付きのブロックレベル XML ファイル（BLXML）に記載する．次に，抽出した Simulink モデル構造情報とコード情報の両方を含むコード情報付き BLXML から，一般的な CSP タスクグラフを生成し，その CSP タスクグラフを介して，モデルのブロック線図に対応する SigmaC のデータフローグラフを生成し，最後に SigmaC ソースコードを生成する．

そこで生成した SigmaC ソースコードは，KALRAY から提供された支援ツールによりデフォルトマッピングを生成し，並列実行できるが，KALRAY のツールに採用されている手法の情報が未公開なので，Kernighan-Lin 法とよばれる既存グラフ分割アルゴリズムを実装し最適化を行った．Kernighan-Lin 法とは，与えられたグラフを二分割し，分割間にノード交換を繰り返し，分割間のエッジカットを最小にするグラフ分割問題を解く効率的な方法である．これを SigmaC に反映すると，クラスタ間の長い NoC 通信回数を少なくし，グラフの１ループの実行サイクル数を減少させることが可能である．

4　評価

評価には共同研究先から提供された， Simulink を用いて設計された車載モータ制御向けの評価モデルを使用した．このモデルにおいて，Triggered subsystem，S-function などの複雑な構造が含まれており，本フローにおいてコード生成とマッピング評価の対象とした．

まず，評価モデルを提案したフローに入力し，生成フローは正しく動作し，共同研究先の環境においても正しく動作できることを確認した．さらに，評価モデルに対し MATLAB Simulink 環境においてシミュレーションを行い，各信号線におけるデータの変化を記録し，生成した SigmaC コードの実機実行のデータと比較し，シミュレーション結果と MPPA での実機実行結果が一致するとわかった．

生成したマッピングを評価するため，KALRAY ツールと実装した KL 法両方を用いて，2 クラスタから 16 クラスタに対するマッピングを生成した．次に，上記の各マッピングを用いて，評価モデルで生成した SigmaC ソースコードを実機実行し，データフローグラフの１ループ毎に実行サイクル数を記録し，サイクル数の平均値，最大値，最小値を測定した．結果を図 1 に示す．評価の結果から，KL 法は，KALRAY ツールが生成するデフォルトマッピングと同等性能が得られるとわかった．

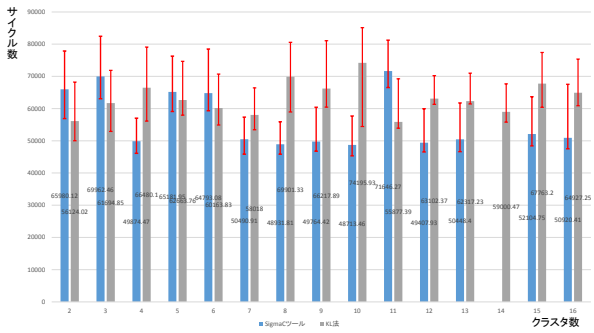

図 1: SigmaC ツールで生成したデフォルトマッピングと KL 法で生成したマッピングの実行サイクル数

5　まとめ

本論文では，MPPA-256 メニーコアを車載制御に用いるため，MATLAB Simulink モデルから KALRAY MPPA-256 向け SigmaC ソースコード生成フローを構築し，生成フローに Kernighan-Lin 法とよばれる既存グラフ分割アルゴリズムを実装し最適化を行った．

実験では，実際のモータ制御モデルを用いて，考案した生成フローで生成した SigmaC ソースコードの動作確認，KL 法より生成したマッピングの評価も行なった．評価結果から，KL 法により，KALRAY ツールと同等性能のクラスタ分割が得られるとわかった．今後は，MPPA におけるタスクの実際の実行時間，通信時間を考慮する分割手法の実装，および他メニーコアにおいて生成フローを適用することなども進める．

参考文献

[1] Kalray S.A. ”MPPA AccessCore 1.0 Introduction to MPPA”.

[2] Kernighan, Brian W., and Shen Lin. ”An efficient heuristic procedure for partitioning graphs.” Bell system technical journal 49.2 (1970): 291-307.

[3] de Dinechin, Benot Dupont, et al. ”A clustered manycore processor architecture for embedded and accelerated applications.” High Performance Extreme Computing Conference (HPEC), 2013 IEEE. IEEE, 2013.