制御性能要求情報に基づいた エンジン制御ソフトの並列性向上手法

福田毅† 入江徹†† 鈴木尊文†† 蛯名朋仁†† 成沢文雄†

厳しい排出ガス規制や更なる燃費向上を達成するため，自動車のエンジン制御に求められる性能は増え続けている．これに対応するため，エンジン制御コントローラにもマルチコアプロセッサの導入が始まっている．しかしながら，エンジン制御ソフトをマルチコア環境向けに並列化する場合，コア間通信するデータ数が多いため，コア間での同期処理が多発しリアルタイム性が損なわれるといった課題がある．そこで本研究では，制御性能要求情報に基づいたエンジン制御ソフト向け並列性向上手法を提案する．提案手法の特徴は，タスク間を跨るデータのディレイ時間と一貫性，同時性の要求情報を基に，同期処理の不要なコア間通信データを特定する点である．本手法を用いて既存のシングルコア用エンジン制御ソフトを，周期タスクを演算処理するコアとエンジン回転に依存したクランク角同期タスクを演算処理するコアへの並列化を実施した結果，約 600 個のコア間通信データのうち 9 割以上のデータが同期不要であることが特定できた．また，並列化した制御ソフトをマルチコアマイコンに実装し，要求されるリアルタイム性能が満たされていることを HILS を用いて確認した．

Parallelism Improvement Method with Requirements of System Control Performance for Engine Control Software

Takeshi Fukuda† Toru Irie†† Takafumi Suzuki†† Tomohito Ebina†† Fumio Narisawa†

The performance requirements of automotive engine control are increasing, for instance to comply the exhaust emission regulations and reduce gasoline consumption. These requirements for engine control system with high performance lead to multicore approaches to reach the desired features. However, the necessary migration of software from single core to multicore systems raises several problems. In particular, an ensuring of real-time performance of engine control software in multicore environment is tough issue due to frequent inter-core synchronizations caused from a large number of inter-core communication data. In this paper, we propose a parallelism improvement method with performance requirements of control for engine control software. The feature of our method is to identify the inter-core data which doesn't need core synchronization according to the requirements of data delay, stability and coherency. We applied our method to legacy engine control software in order to migrate to multicore which has two cores. The first core executes periodic tasks, and the second core executes event tasks which are triggered by an engine rotation. The result indicated that more than ninety percent data out of approximately six hundreds inter-core communication data doesn’t need synchronization mechanism. We evaluated the parallelized engine control software with HILS(Hardware-in-the loop simulation), the software satisfied requirements of real-time performance.

†(株)日立製作所
Hitachi, Ltd.
†† 日立オートモティブシステムズ(株)
Hitachi Automotive Systems, Ltd.

1. はじめに

環境問題とエネルギー危機の解決に向け，自動車業界でも排気ガス中の CO_2 や HC, NOx といった大気汚染物質の削減が求められている．自動車排出ガスへ

の規制は世界中で強化され続けており(図 1), 2014 年より欧州で適用が開始された Euro6 規制では新たに直噴ガソリン車への PM(Particulate Matter)粒子数にも規制が制定された[1].

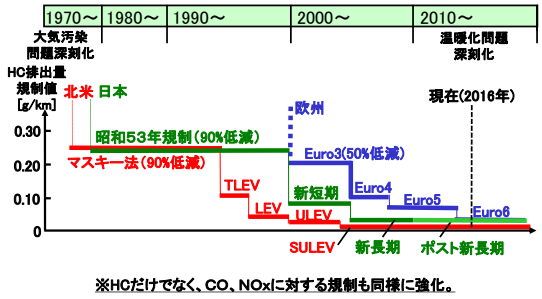

図 1 排出ガス規制(HC)の強化動向

これら排出ガス規制に対応しつつ燃費効率を向上させるため, 近年, 自動車エンジンへの排気再循環(Exhaust Gas Recirculation)機構の新規導入や, 電子制御による一層のエンジン燃焼制御の効率化が進んでいる. これに伴いエンジン制御ソフトのプログラム規模が増加し続けており, エンジン ECU(Electronic Control Unit)に必要とされる CPU 性能も増え続けている[2].

従来はエンジン ECU の CPU クロック周波数を上昇させることで単位時間当たりの処理性能向上を実現してきた. しかしながら, 消費電力の増大や発熱量の問題からこれ以上のクロック数上昇は困難となったため, 近年は自動車分野でもマルチコアを用いた並列処理による性能向上が導入され始めた[3]. マルチコアマイコンの性能を活かすには, 既存のシングルコア用制御ソフトをマルチコア向けに並列化する必要がある.

古くよりソフトの並列化を目的としたスケジューリングアルゴリズムの研究がなされ, DAG(Directed Task Graph)を用いた静的なスケジューリング手法についても数多く報告がある[4]. 近年はリアルタイム性が強く求められるディーゼルエンジン制御ソフトを対象とした並列化手法についても報告がある[5].

しかしながら, データ依存関係が複雑なエンジン制御ソフトをマルチコア向けに並列化する場合, コア間通信データ数が多いためコア間で同期処理が頻発し, 同期待ち時間が増大するためリアルタイム性が損なわれるといった課題がある.

そこで本研究では, 制御性能要求情報に基づいた制御ソフト向け並列性向上手法を提案する. 提案手法の特徴は, 許容データディレイ時間とデータ一貫性, 同時性の要求情報を基に, 同期処理の不必要なコア間通信データを特定する点である.

本手法を用いて既存のシングルコア用エンジン制御ソフト一式を, 周期タスクを演算処理するコアと, エンジン回転に依存したクランク角同期タスクを演算処理するコアへの並列化を実施した結果, 約 600 個のコア間通信データのうち 9 割以上のデータが同期不要であることが特定できた. また, 並列化したエンジン制御ソフトをマルチコアマイコンに実装し, HILS (Hardware-in-the-loop simulation)を用いて動作検証したところ, 並列化後の制御ソフトが要求されるリアルタイム性能を満たしていることが確認できた.

本紙の貢献は次の 2 点である.

1. リアルタイム性が強く要求される制御システム向けの並列性向上手法として, 制御性能要求情報に基づいてコア間同期処理が不要なデータを特定することで, 同期処理による待ち時間を削減可能であることを提案した.
2. 制御性能要求情報を基に並列化することで, タスク間に依存関係(書込み／読込み処理)がある場合でも並列動作が有効であることを示した点.

以降, 2 章では自動車制御ソフトのマルチコア移行技術に関する関連研究, 3 章では本研究で検討するタスク単位での並列化について, 4 章では並列化対象の自動車エンジン制御ソフトの概要, 5 章では提案するエンジン制御ソフトの並列性向上手法, 6 章では提案手法を既存のシングルコア向けエンジン制御ソフト一式に適用した結果, 7 章では提案手法適用結果に対する評価と考察について示し, 8 章で結論を述べる.

2. 関連研究

シングルコア向けの車載用制御ソフトをマルチコア環境へと効率的に移行させることを目的に, 自動車ドメインで提供されているマルチコア CPU およびマルチコア対応 OS の調査結果と, レガシーソフトのマルチコア移行ガイドラインが提示されている[6]. ガイドラインでは, マルチコア移行時の大きな問題の一つとしてマルチコアへのシステム分割の決定問題, いわゆるコア割当て問題が取り上げられている.

マルチコアへのコア割当て問題に対しては, 自動車分野でも既に様々な手法が提案されている. これらの手法はシステム単位やコンポーネント単位, インストラクション単位といった並列化の粒度に応じて大別できる[7].

並列化粒度が最も大きい例では, 複数の車載システムを 1 つのマルチコア ECU 上で動作させるシステム

単位の並列化手法が提案されており，コアへの割当てをシステム間の依存関係を制約条件とした組み合わせ最適化問題と捉え，SMT ソルバを用いて解く手法が知られている[8]．この場合，従来は複数個の ECU を用いて実現していた機能を一つのマルチコア ECU に統合可能なため，自動車の総 ECU 数を削減できるといった効果がある．

ECU 数の削減を目的としたシステム単位の並列化に対し，エンジン制御機能といった単体のシステムの性能向上を目的とした並列化手法も数多く提案されている．特に単体システムの並列化では，SW-C（Software Component）や C 言語の関数ごとにコアに割り当てるコード単位の並列化手法と，SIMD（Single Instruction Multiple Data）を例とするインストラクション単位の並列化を考慮した手法が提案されている[9]．

単体システムの具体的な並列化手法としては，例えば制御ソフトを CSP（Communicating Sequential Processes）理論に基づいた C コードへと変換し，コアへのタスク割当てを非線形問題として解く手法が提案されている[10]．また，制御ソフト向けに階層的に並列化処理を行うマルチグレイン並列化処理を特徴とする並列化コンパイラ OSCAR による並列化手法が知られている[11]．

これら単体システム向け並列化手法の多くは，1 つの OS タスク内を対象としたソフト並列化手法である．このため，複数の OS タスクを持つ制御システムでは，それぞれの OS タスクの開始時と終了時にコア間同期処理を挟む手法が提案されている[12]．これにより，制御ソフト全体をマルチコア上で動作させることが可能となる．

しかしながら，エンジン制御ソフトのような複数のプリエンプティブタスクと多くの割込み処理を持つ制御ソフトに既存手法を適用した場合，優先度スケジューリングによりコア毎の処理時間が大きく変動するため，コア間同期のための待ち時間が増大するといった課題がある．また，関連研究の多くは，既存の制御ソフトのデータ依存関係とプログラムの先行制約条件を基に並列化する手法であるため，エンジン制御ソフトのような，複数のタスク間で複雑なデータ依存関係を持つ制御ソフトに適用した場合，コア間での同期処理が頻発するといった課題がある．このため，制御ソフト向けにコア間同期処理を削減する手法の開発が必要とされている[13]．

3. 並列化粒度

ここでは，従来研究に多い 1 つの OS タスクを対象としたタスク内の並列化と，本研究で検討対象とするタスク単位での並列化を制御ソフトに適用した場合の違いについて述べる．

3.1. タスク内並列化

図 2 に，従来研究の多くが検討対象としているタスク内並列化手法を用い，10ms タスクを 2 つのコアを持つ ECU 向けに並列化した場合の概念図を示す．制御ソフトはデータ（制御信号）の依存関係が強く，図右のようにコア間の同期処理が多発する．

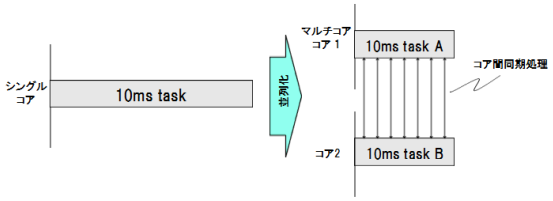

図 2 タスク内並列化の概念図（1 タスク）

さらに，図 2 の並列化結果に，優先度の高い周期タスク（1ms タスク）と割込み処理を 1 つずつ追加した場合の概念図を，図 3 に示す．

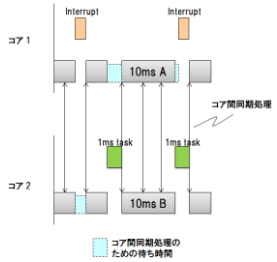

図 3 タスク内並列化の概念図（複数タスク）

組込み制御システムはリアルタイム性の確保を目的に，優先度スケジューリング手法を採用していることが多い．このため，優先度の低いタスクは優先度の高いタスクや割込み処理により処理が一時停止される．この結果，並列化したタスク間でコア間同期処理のための待ち時間が増大し，全体としてのリアルタイム性が損なわれると言った課題がある．これは，組込み制御システムがその厳しいコスト制約により，コア数の少ない ECU を採用しいていることも要因となっている．

3.2. タスク単位並列化

図 4 に，タスク内並列化（図左）とタスク単位並列化（図右）の概念図を示す．タスク単位並列化の利点は，

タスク内並列化と比較してコア間の同期処理が少ない点である．このため，同期処理のための待ち時間の削減が期待できる．

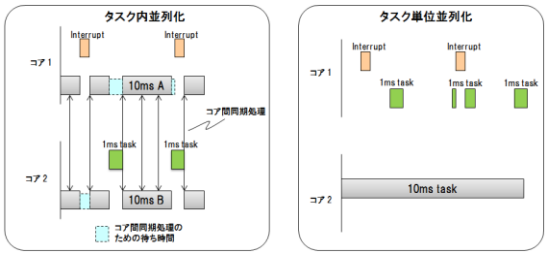

図 4 タスク内並列化とタスク単位並列化

以上のことから，従来研究の多くが 1 つの OS タスク内の並列化を対象としていたことに対し，本研究では，OS タスク単位での並列化を検討対象とする．

4. エンジン制御ソフトの概要

厳しい排出ガス規制や燃費効率の向上に対応するためエンジン制御の電子化は進み続けており，近年のエンジン制御ソフトのプログラム量は 150 万行とも言われている[3]．ここでは，本研究が対象とするエンジン制御ソフトの特徴を述べる．

4.1. 周期タスクと割込み処理

多くの制御ソフトと同じくエンジン制御ソフトも，センサからの入力信号を基に操作量を演算し，結果をアクチュエータへと出力することでエンジンを制御している．より具体的には，エンジン ECU はエアフローセンサや温度センサといった物理量やクランクシャフトの角度センサ信号を入力とし，燃料噴射量および点火タイミングを演算する．そして，演算結果を基にインジェクタやイグナイタを動作させることでエンジンを制御している．

これら一連のシステム制御を実現するため，エンジン制御ソフトは数十の周期タスクと，割込みイベントをトリガに演算処理される百前後の割込み処理を持つ[3]．

4.2. データ依存関係

自動車エンジンは複数個のセンサ入力と複数個のアクチュエータ出力が互いに影響し合うため，データ（制御信号）は複雑に演算処理されている．実際に，全データ中の約 70%がモジュール間で利用されており，エンジン回転スピードや空気温度といったデータにいたっては 100 以上のモジュールで利用されているといった報告がある[3]．

複雑なデータの依存関係（書込み／読込み処理）はモジュール間だけでなく，前述の周期タスクと割込み処理といったタスク間でも存在している[12]．そして，エンジン ECU への高いパフォーマンス要求値を満たすため，これらのデータのほとんどがグローバル変数を用いてデータ共有されている[3][14]．

4.3. データフローとタイムチャートの例

図 5 に，エンジン制御システムの一機能である燃料噴射制御のデータフローの例を示す．

図の場合，燃料噴射制御を実現するために，1 つの外部入力信号と 3 つのタスク処理が関係している．最初に，(1)エアフローセンサ情報が外部より入力される．次に，(2)エアフローセンサ情報を元に，空気量を演算し特定する．最後に，空気量情報をもとに(3)燃料噴射量と(4)噴射タイミングを決定する．また，燃料噴射タイミングは燃料噴射量にも依存するものとする．

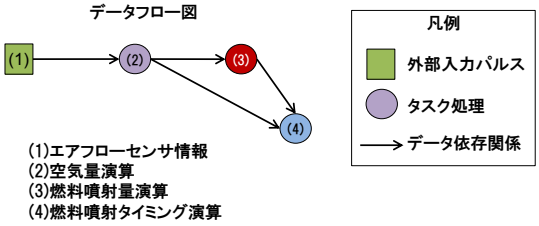

図 5 燃料噴射制御のデータフロー例

図 6 に，上述した燃料噴射制御のタイムチャートの例を示す．図の場合，外部入力パルス信号であるエアフローセンサ情報は，500 μs 毎に入力されるものと考えている．また，3 つのタスク処理はそれぞれ 2ms と 4ms の周期処理タスクと，エンジン回転信号による割込み処理として実現される場合を想定している．

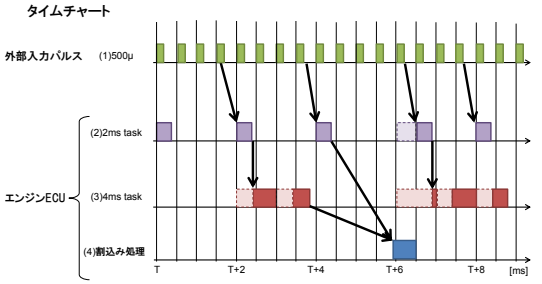

図 6 燃料噴射制御のタイムチャート例

前述のデータフローの通り，それぞれの処理にはデータ依存関係があるため，タイムチャートでは 4 つの処理間にまたがるデータ依存関係を矢印で示している．

シングルコアマイコンでは，2 つの周期タスクと 1 つの割込み処理が同じプロセッサコア上で動作するため，優先度スケジューリングによりそれぞれのタスクの優先度に応じて演算処理されるタスクが決定する．一方マルチコアマイコンを用いた場合，複数のコアで別々のタスクを並列処理可能なため，制御ソフト全体の処理完了時間を短縮することが可能となる．

5. 提案手法

既存のシングルコア向け制御ソフトをマルチコア環境で動作させる場合，コア間通信データの同期処理が頻発するためリアルタイム性が損なわれるといった課題がある．本章では，制御性能に対する要求情報である許容データディレイ時間と入力データセットの要求情報に基づき，コア間通信データを分類する提案手法について述べる．

5.1. 全体像

提案手法は次の 3 ステップに分かれている(図 7)．第 1 のステップでは，プログラム解析手法を用いてデータの依存関係を解析する．第 2 のステップでは，データ依存関係とコア割当て方針から，コア間通信データを特定する．第 3 のステップでは，コア間通信データを対象に，許容データディレイ時間と入力データセットの要求情報から，同期処理が不要なコア間通信データを分類する．

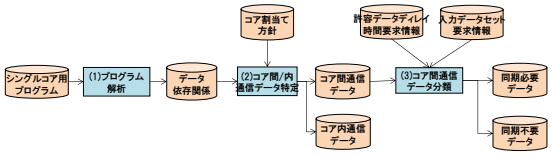

図 7 提案手法の全体像

5.2. プログラム解析

既存のシングルコア用制御ソフトを入力とし，プログラム解析によりデータの依存関係を抽出する．ここで，前述の通りエンジン制御ソフトはパフォーマンスの関係からグローバル変数を多用しているため，これに対応したプログラム解析が必要となる．

5.3. コア間/内通信データの特定

データ依存関係とコア割当て方針から，制御信号をコア間通信データとコア内通信データとに分類する．ここで，本研究が検討対象とするタスク割当て方針とは，OS タスク単位でのコア割当てを想定している．

5.4. コア間通信データの分類

制御ソフトの周期タスクやイベント処理は，一定の時間やイベントの度に演算処理されるといった特徴がある．このため，全てのコア間通信データにコア間同期処理が必要なわけではなく，要求される性能値が満たされる場合に限っては最新値以外の演算処理結果を利用することで，コア間同期処理を省くことが可能である．提案手法では，制御性能要求情報である許容データディレイ時間および入力データセットの要求情報から，同期処理が不要なコア間通信データを特定する．

(a)許容データディレイ時間による分類

本紙でのデータディレイ時間とは，エンジン ECU の外部入力/出力信号から任意のタスク演算処理開始/完了までの時間を指す．このデータディレイ時間に対する許容時間情報から，コア間同期処理が必要かどうかを判断する．

図 8，図 9 に，前章で紹介したエンジン制御ソフトの一例を基にした入力信号のデータディレイ時間の例を示す．図では，2ms タスクをコア 1 に，4ms タスクをコア 2 に割り当てた例を示している．ここで，提案手法におけるデータディレイ時間とは，AD 変換器による入力信号変換から，2ms タスクの演算処理を挟み，4ms タスク処理が開始するまでの時間を指す．

2ms タスクと 4ms タスクの間でコア間同期処理を実施する場合を，図 8 に示す．コア間同期処理を実施する場合，2ms タスク演算完了直後に 4ms タスクが演算開始するため，データディレイ時間(図中の青矢印)を短くすることができる．しかしながら，コア 2 側にコア 1 側の演算処理が完了するまでの待ち時間が発生するため，マルチコアの利点である並列処理が実現できていない．

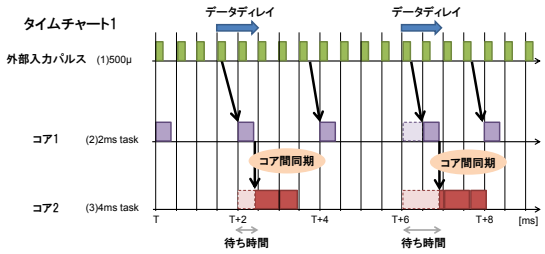

図 8 コア間同期処理が有る場合

一方，コア間同期が無い場合(図 9)は，コア間での待ち時間が生じないためマルチコアの並列処理が活かせる．ただし，同期がある場合に比べ，4ms タスクは最新の 2ms タスク以外のデータを利用するため，データ

ディレイ時間が長くなる点に注意する必要がある.

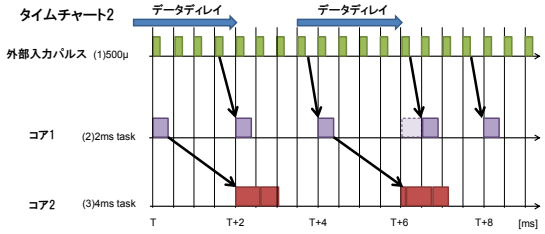

図 9 コア間同期処理が無い場合

(b)入力データセットによる分類

本紙での入力データセットとは，タスク内で読込み処理されるデーター式を指す．入力データセットによる分類は，さらにデーター貫性による分類とデータ同時性による分類の 2 つがある．

(b-1)データー貫性による分類

データー貫性による分類とは，タスク演算処理中における最新値データの要否に基づいた分類である．

図 10 に示す例の場合のように，4ms タスク処理中に 2ms タスクが 2 回演算処理を完了する場合がある．この時，2 回目の 2ms タスクの出力結果を 4ms タスクに反映すべきか否かは，その制御アルゴリズムに依存する．4ms タスクが 2ms タスクの最新値データを必要とする場合は，コア間同期処理が必要となる．

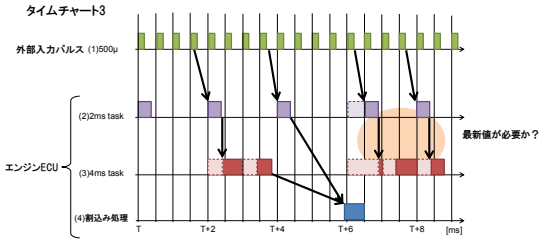

図 10 データー貫性の要否判断

(b-2)データ同時性による分類

データ同時性による分類とは，タスクの入力データ間でのタイミングに基づいた分類である．

図 11 に示す例の場合，割込み処理は 2ms タスクと 4ms タスクの両方の出力データを入力としている．この時，割込み処理が，4ms タスクで入力として利用された 2ms タスクの出力データを必要とする場合と，最新の 2ms タスクの出力データを必要とする場合とがある．

割込み処理が，4ms タスクと同じタイミングの 2ms タスクの出力データを必要とする場合は，割込み処理と 2ms タスクの間でコア間同期処理の必要が無いと判断できる．

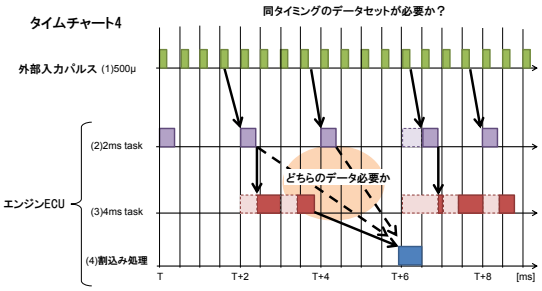

図 11 データ同時性の要否判断

6. エンジン制御ソフトへの適用

本章では，提案手法を既存のシングルコア用エンジン制御ソフトに適用した結果について述べる．

6.1. 適用対象ソフトと想定するマルチコア環境

今回適用対象としたエンジン制御ソフトは，シングルコア向けエンジン制御ソフト一式で，制御信号数は約 10,000 個ある．

また，マルチコア環境は自動車エンジン向けマルチコア ECU（2 コア）を想定し，タスクのコア割当ては，周期タスクを演算処理するコアと，エンジン回転に依存したクランク角同期タスクを演算処理するコアの2つに分けることを想定した．

6.2. プログラム解析結果

プログラム解析ツールを用いて，エンジン制御ソフト一式からデータの依存関係情報を抽出した．図 12 に，C 関数間のデータ依存関係解析結果の一部を示す．図の通り，エンジン制御ソフトはデータ依存関係が複雑で，制御信号が複雑に影響し合っていることが分かる．

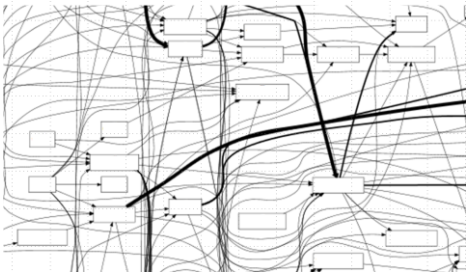

図 12 データ依存関係の抽出結果（一部抜粋）

6.3. コア間通信データの特定結果

データ依存関係の解析結果と想定するコア割当てから，コア間通信データを特定した．図 13 に，コア間/内通信データの特定結果の一部を示す．解析の結果，

約 10,000 個の制御信号のうち約 600 個のコア間通信データを特定することができた.

Variable \ Type	CORE1	CORE2	コア間／コア内判定
variable_231	R/W	W	コア間
variable_232	R	R/W	コア間
variable_233	R/W	R	コア間
variable_234	-	W	コア内
variable_235	R	R/W	コア間
variable_236	R	R/W	コア間
variable_237	-	R/W	コア内
variable_238	R/W	W	コア間
variable_239	R/W	W	コア間
variable_240	R/W	R	コア間
variable_241	R/W	R	コア間
variable_242	W	R	コア間
variable_243	-	R/W	コア内
variable_244	-	R/W	コア内
variable_245	-	R/W	コア内

図 13 コア間通信データの特定(一部抜粋)

6.4. コア間通信データの分類結果

コア間通信データ約 600 個を対象に, 制御性能要求情報である許容データディレイ時間と入力データセット情報を元に, 人手によりコア間同期処理が必要なデータと不要なデータとに分類した. 図 14 に, データ分類の結果を示す. 今回のコア割当ての場合, コア間通信データ約 600 個のうち, 9 割以上のデータが同期不要であることがわかった.

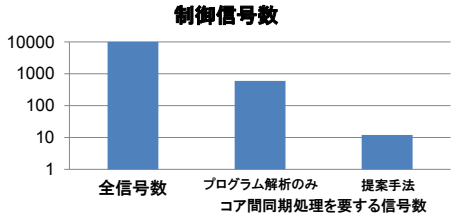

図 14 データ分類結果

6.5. コア間通信手法

コア間同期処理が不要なデータについては, CABs(Cyclic Asynchronous Buffers)[15]による実装を行った. CABs は起動周期が異なる処理間のデータ通信に好適な 1:N 通信手法で, データの書込み処理と読込み処理が互いにブロッキングし合うことなく動作可能となっているため, リアルタイムシステム一般に効果的な通信手法である.

7. 評価と考察

提案手法の評価を目的に, 既存のシングルコア用エンジン制御ソフト一式を並列化し, エンジン向けマルチコア ECU に実装した. 本章では, 並列化したエンジン制御ソフトを搭載したマルチコア ECU を対象に, HILS を用いて動作検証した結果について述べる.

また, 更なる高並列化の実現を目的に, 今回採用したタスク単位並列化に加え, 更にタスク内並列化を組み合わせた場合の並列化評価について述べる.

7.1. HILS 評価

図 15 に, HILS を用いて測定した CPU 負荷率の計測結果を示す. 図の横軸はエンジン回転数[rpm]を示しており, 縦軸はそれぞれのプロセッサコアの CPU 負荷率を示している.

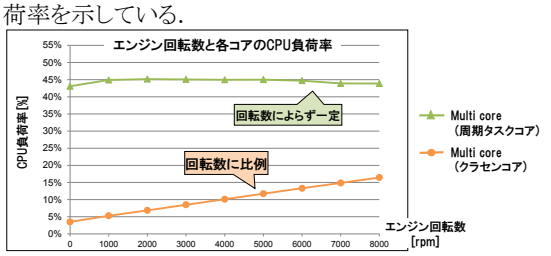

図 15CPU 負荷計測結果

図の通り, 周期タスクを割り当てたコアはエンジン回転数によらず一定の CPU 負荷率となっていることがわかる. 一方で, エンジン回転に依存したクランク角同期タスクを割り当てたコアは, エンジン回転数に比例して CPU 負荷率が上昇していることがわかる. このことから, それぞれのコアが制御周期タスクと回転角同期タスクを並列処理していることが確認できた.

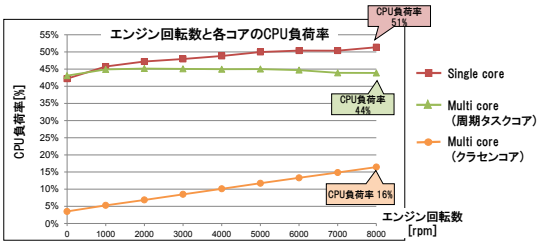

図 16 シングルコアとの CPU 負荷率の比較

図 16 にシングルコアとの CPU 負荷率の比較を示す. 図より, 提案手法を用いてエンジン制御ソフトを並列化しマルチコア ECU 上で演算処理することにより, CPU 負荷率を各コアに分散できることが確認できた.

さらに, 並列化後のエンジン制御ソフトの応答性能を確認するため, HILS を用いてエンジンの燃料噴射処理と点火処理の信号出力を確認した結果, それぞれが要求される応答時間を満たしていることが確認できた. このことから, 提案手法を用いることで, シングルコア向け制御ソフトからリアルタイム性能を満たしたマルチコア向け制御ソフトを作成可能であることが確認できた.

7.2. タスク内並列化との組合せ

表 1 に，今回実施したタスク単位での並列化の HILS 結果と，今回の並列化に加え，さらに最も処理時間が長かった 1 つの周期タスクをタスク内並列化した場合のシミュレーション（HW 無し）結果について記す．尚ここで，タスク内並列化は処理完了時間を最小となるようタスク内を並列化したもので，表の Core 負荷率はエンジン回転数 8000rmp 時を想定したものである．

表 1 並列化結果の比較

並列化粒度	タスク単位	タスク単位＋タスク内
測定方法	HILS	シミュレーション(HW 無し)
Core1 負荷率	44%	39%
Core2 負荷率	16%	21%
コア間通信データ数	約 600 個	約 1800 個

表に示す通り，今回検討したタスク単位並列化に加え更にタスク内並列化を組み合わせることで，CPU 負荷率については更なる分散平準化が可能な見込みが得られた．

しかしながら，タスク内並列化はコア間通信データ数も大幅に増加するため，リアルタイム性の確保に課題が残る．また，自動車分野では製品のマイナーチェンジごとに制御ソフトに一部変更が入ることが多いが，タスク内並列化を採用した場合はこの一部変更による影響が複数のコアに波及する可能性があるため，マイナーチェンジごとに多くの検証工数が必要となる点にも課題がある．

7.3. 考察

提案手法により 9 割以上ものコア間通信データが同期処理不要だった理由として，本研究の前提条件としたタスク単位でのコア割当てが大きく影響していると考える．1 つの OS タスク内でのコア割当てを検討する場合は，さらにタスク内の制御信号に対し，データディレイ時間の要求情報が必要となると想定している．

エンジン制御ソフトは割込み処理が多いため，コア間同期処理時の同期待ち時間の見積もりが難しい．これを回避する方法として，コア数の多いメニーコアを利用することが検討される．例えば，割込み処理を一切割り当てない任意のコアを用意することで，特定のコアに関しては割込み処理に起因する応答時間の変動を排除することが可能になる．これにより，コア間の同期待ち時間を平準化できると想定している．これは，特に 1 つの OS タスク内のソフトを並列化する場合に効果的と考える．

今後，マルチコアまたはメニーコアを用いてエンジン制御ソフトを並列に動作させるにあたり，従来のシングルコア向け開発では定義されていなかった新たな要求仕様の項目として，本研究でのデータディレイ時間およびデータ一貫性・同時性の要求値を規定することにより，より並列効果の高いコア割当てが可能になると考える．

この要求仕様の策定については，制御設計時に要求値を決定するトップダウンなアプローチによる研究が開始されている[16]．また，並列動作する制御ソフトへのこれら要求やデッドライン保障について，並列後の制御ソフトで正しく満たされているか検証する手法についても取組みが始められている[17]．

8. おわりに

本研究では，複数のタスクにデータ依存関係がまたがったエンジン制御ソフトを対象に，制御ソフトの並列性を向上させる手法を提案した．提案手法を既存のシングルコア向けエンジン制御ソフトに適用した結果，約 600 個のコア間通信データのうち 9 割以上のデータが同期不要であることを特定した．

また，並列化したエンジン制御ソフトをマルチコア ECU に実装し，HILS を用いて動作検証した結果，並列化後の制御ソフトが要求されるリアルタイム性能を満たしていることが確認できた．

今後の課題として，同期処理不要なコア間通信データ判定の自動化がある．本点については，マルチコア向けタイミング解析ツールを用いることで解決を図る．また，タスク単位での最適なコア割当ても，重要な課題の一つである．

謝辞 本研究を進めるにあたり，自動車エンジン制御ソフトへの提案手法の適用にご協力いただいた日本プロセス株式会社の仲尾次達哉殿，徳田佳秀殿，佐藤晃司殿に感謝する．

参考文献

[1] Delphi: Worldwide Emissions Standards, http://delphi.com/docs/default-source/catalogs/del

phi-worldwide-emissions-standards-pc-ldv-15-16.pdf, 2015.
[2] R. Mader, A. Graf and G. Winkler, AUTOSAR Based Multicore Software Implementation for Powertrain Applications, SAE World Congress & Exhibition, Apr. 2015.
[3] L. Michel, T. Flaemig, D. Claraz and R. Mader, Shared SW development in multi-core automotive context, Embedded Real Time Software and Systems (ERTS2'16), Jan. 2016.
[4] Y. Kwok and I. Ahmad, Static Scheduling Algorithms for Allocating Directed Task Graphs to Multicoreprocessors, ACM Computer Surveys, Dec. 1999.
[5] S, Kehr, E, Quiñones, B. Böddeker and G, Schäfer, Parallel Execution of AUTOSAR Legacy Applications on Multicore ECUs with Timed Implicit Communication, Design Automation Conference (DAC'15), Jun. 2015.
[6] G. Macher, A. Höller, E. Armengaud and C. Kreiner, Automotive Embedded Software: Migration Challenges to Multi-Core Computing Platforms, International Conference on Industrial Informatics (INDIN'15), Jul. 2015.
[7] R. Schneider, D. Juergens and A. Kohn, Software Parallelization in Automotive Multi-Core Systems, SAE World Congress & Exhibition, Apr. 2015.
[8] W. Schwitzer, R. Schneider, D. Reinhardt and G Hofstetter, Tackling the Complexity of Timing-Relevant Deployment Decisions in Multicore-Based Embedded Automotive Software Systems, SAE World Congress & Exhibition, Apr. 2013.
[9] J. Castrillon, L. Thiele, L. Schorr, W. Sheng, B. Juurlink, M. Alvarez-Mesa, A. Pohl, R. Jessenberger, V. Reyes and R. Leupers, Multi/Many-Core Programming: Where are we Standing?, Design, Automation & Test in Europe Conference & Exhibition (DATE'15), Mar. 2015.
[10] 大川禎, 枝廣正人, 久村孝寛, CSP 理論にもとづいた制御モデルのマルチコア実装向けタスク割当て, 組込み技術とネットワークに関するワークショップ(ETNET), 2013 年 3 月.
[11] Y. Kanehagi, D. Umeda, A. Hayashi, K. Kimura and H. Kasahara, Parallelization of Automotive Engine Control Software On Embedded Multi-core Processor Using OSCAR Compiler, Cool Chips XVI, Apr. 2013.
[12] M. Panić, S. Kehr, E. Quiñones, B. Boddecker, J. Abella and F. J. Cazorla, RunPar: An Allocation Algorithm for Automotive Applications Exploiting Runnable Parallelism in Multicores, International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS'14), Oct. 2014.
[13] 福田毅, 入江徹, 鈴木尊文, 蛯名朋仁, 成沢文雄, プログラム解析を用いたエンジン制御ソフトのマルチコア移行手法, 組込み技術とネットワークに関するワークショップ(ETNET), 2016 年 3 月.
[14] S. Fürst, An OEM's point of view on multi-core, Embedded Multi-Core Conference (EMCC'15), Jun. 2015.
[15] G. Buttazzo, Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications – Second Edition, Springer, 2005.
[16] C. Ebert and T. Flaemig, Software development in collaboratively engineered systems –Development from single-core, multi-core, many-core to distributed Hardware, Embedded Multi-Core Conference (EMCC'16), Jun. 2016.
[17] J. Härdtlein, Distributing automotive real-time systems, Embedded Multi-Core Conference (EMCC'16), Jun. 2016.