システムソフトウェアとオペレーティング・システム 86−10
(2001. 3. 6)

RMA 手法のエンジン制御システムへの適用に関する研究

飯山 真一　遠藤 友悟　高田 広章
豊橋技術科学大学 情報工学系
〒 441-8580 豊橋市天伯町雲雀ヶ丘 1-1
{shin,yugo,hiro}@ertl.ics.tut.ac.jp

菅沼 英明
トヨタ自動車 第 2 電子技術部
〒 410-1193 裾野市御宿 1200
hideaki@suganuma.tec.toyota.co.jp

あらまし　自動車のエンジン制御システムは, 燃費向上や排気ガスのクリーン化の要求からソフトウェアの大規模化・複雑化が著しく, 体系的なソフトウェア設計手法の導入が求められている。本研究では, タスクモデルとしてマルチフレームタスクモデルを採用し, Rate Monotonic Analysis(RMA) の手法を自動車のエンジン制御システムへの適用を目指している. そこで, 実車でエンジン制御システムの各タスクの最大実行時間を計測し, 得られたデータを用いて, 必要十分条件によるスケジュール可能性解析を行った. スケジュール可能性解析では, 一般化されたマルチフレームタスク (GMF タスク) からなるタスクセットのスケジュール可能性を必要十分条件を効率的にチェックする方法として提案されている, maximum interference function(MIF) の概念を用いた. 本稿では, エンジン制御システムへの RMA 手法の適用と, そのために開発したスケジュール可能性解析手法の適用事例を報告する.

キーワード　スケジュール可能性解析 エンジン制御システム, rate monotonic analysis(RMA), マルチフレームタスクモデル, maximum interference function(MIF)

Applying RMA to an Engine Management System

Shinichi Iiyama　Yugo Endo　Hiroaki Takada
Dept. of Information and Computer Sciences,
Toyohashi Univ. of Technology
1-1 Hibarigaoka, Tempaku-cho, Toyohashi 441-8580, Japan
{shin,yugo,hiro}@ertl.ics.tut.ac.jp

Hideaki Suganuma
Electronics Engineering Div. II
Toyota Motor Corporation
1200 Mishuku, Susono 410-1193, Japan
hideaki@suganuma.tec.toyota.co.jp

Abstract　The software of engine management systems are getting larger and more complex rapidly, in order to meet requirement for fuel efficiency and low exhaust. And so some systematic software design approachs are required. In this study, we measures the maximum execution time of each task using a tool which is used as an on-board evaluation environment of an engine management system, and tests the utilization bound test and with a necessary and sufficient test. We adopted generalized multiframe task model and the maximum interference function(MIF) which checks the condition efficiently for schedulability test. In this paper, we describe that applying RMA to an engine management sytem and developed schedulability analysis method.

Key words　schedulability analysis, engine management system, rate monotonic analysis(RMA), multiframe task model, maximum interference function(MIF)

1 はじめに

近年, 自動車のエンジン制御システムは, 走行性, 低燃費化や排気ガス規制などの要求からソフトウェアの大規模化・複雑化が著しい. また, エンジン制御を含めた複数の制御を 1 つのプロセッサで実現する複合化の動きや, 自動車内のネットワークを用いて他のシステムとの連携動作させるケースも増えつつある.

ソフトウェアが大規模化する中で, ソフトウェアの品質と生産性を高く保つために, ソフトウェア開発の効率化を支援するツールの採用や, 体系的なソフトウェア設計手法の導入が求められている.

近年, エンジン制御システムにおいてもリアルタイム OS が採用される傾向にあり, これにより, 制御を機能的な振舞いと時間的な振舞いとに分離することが可能となっている. このうち, 時間的な振舞いを検証するために, Rate Monotonic Analysis(RMA) などのリアルタイムスケジューリング理論の導入が不可欠となる.

本研究では, タスクモデルとしてマルチフレームタスクモデル [1] を採用し,RateMonotonic Analysis(RMA) の手法を自動車のエンジン制御システムへの適用を行った. 実車を用いてエンジン制御システムの各タスクの最大実行時間を計測し, 得られたデータを用いて, 必要十分条件によるスケジュール可能性解析を行った. スケジュール可能性解析では, 一般化されたマルチフレームタスク (GMF タスク) からなるタスクセットのスケジュール可能性を必要十分条件を効率的にチェックする方法として提案されている,maximum interference function(MIF) の概念を用いた.

本稿では, エンジン制御システムへの RMA 手法の適用と, そのために開発したスケジュール可能性解析について述べる. 以下,2 節において, 本研究で解析対象とするエンジン制御システムの概略構成を紹介した後,3 節において動作ログの取得方法とその解析方法について述べる.4 節では, 解析で得られた結果を用いて,RMA 手法を適用したスケジュール可能性解析の結果について述べる.

2 エンジン制御システム

自動車のエンジン制御は, おおよそ, エンジン回転数やアクセル開度などのセンサから得られる入力信号を処理し, 燃料の噴射量, 噴射時間や噴射タイミングなどを決定し, 各アクチュエータに出力することで実現されている. センサからの情報を処理し, アクチュエータを制御するために, ECU(Electronic Control Unit) と呼ばれるマイクロコントローラが使用されており, その中には,1 つまたは複数のプロセッサや RAM,ROM の他にセンサやアクチュエータとのインターフェース回路が収められている. ROM には, エンジン制御の処理手順を記述した ECU ソフトウェアと呼ばれるプログラムが収められている. この ECU ソフトウェアは, システムの品質や信頼性を決定づける重要な役割を果たしている.

エンジン制御を行う処理は, 一定の時間周期で起動される時間同期型の処理とエンジンの回転に同調して一定の回転角周期で起動される回転角同期型の処理に分けられる. 具体的に, 前者には冷却水の温度センサからの信号から各種の補正パラメータを計算する処理などが含まれ, 燃料の噴射タイミングを決定する処理などは後者に含まれる.

多くのエンジン制御システムには,RAM と ROM を内蔵し外部にバスを出していないワンチップマイコンを用いられているため, システムで使用できるメモリ量に厳しい制限がある. エンジン制御を行うために非常に多

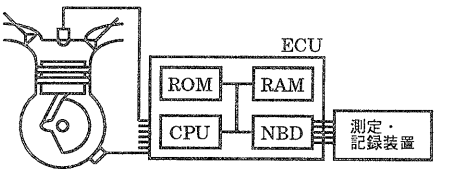

図 1: エンジン制御システムの概要

くのセンサやアクチュエータが使用されているため, 制御を行う個々の処理の数も極めて多いが, 各処理を別々のタスクで実現することは, 限られた RAM/ROM に収めることができなくなるばかりでなく, 処理時間のオーバヘッドの増加を招く. そこで, 各処理を起動周期や相対デッドラインごとにタスクを設け, 機能単位ごとにタスク分割を行っている.

本研究で対象としているシステムには, ワンチップマイコン内に NBD(Non-Break Debug block) と呼ばれる機構を持たせている. NBD は, マイコン外部から, プロセッサの動作に影響を与えずにマイコン内部のメモリを読み書きするための機構である [2]. システム評価時には, この NBD の機構を用いて, あらかじめ指定したメモリ領域を周期的に読み込み, 読み込んだデータを表示・記録する装置 (以下, 記録装置) を用いる.

NBD と記録装置は, 元来, エンジン制御システムの動作状態をエンジンを動かしながら監視することで, 各種の制御パラメータの最適化を行うための機構で, 本研究で行っているタスク切替えなどの離散的なイベントを記録するためのものではない. そのため, これらの機構を用いて取得・記録できるデータ量は極めて限られている. 具体的に本研究では, 記録装置の読み込み周期が 1ms で,16bit のデータを最大 48 個の記録という制限が課せられている.

3 システム動作の解析

3.1 イベント情報の取得

システムの時間的振舞いを調べるためには, (1) タスク切替え,(2) タスク起動/起床,(3) タスクの終了/スリープ, (4) 割込みハンドラの起動/終了に関する情報を区別して記録することが望ましい. しかし前述したように現状では, 記録装置により取得できるデータ量に制約があるため, 取得する情報を制限せざるを得ない.

そこで, 本研究では, タスク切替えと割込みハンドラの起動に関する情報に的を絞り, これらの情報をメモリ上に記録するように, リアルタイム OS の改造を行った. 我々は, タスク切替えと割込みハンドラの起動をまとめてイベントと呼び, それに関する情報をイベント情報と呼んでいる. イベント情報はイベントが発生するとできる限り早くメモリ上に記録され, メモリ上に記録されたイベント情報は NBD を経由して記録装置記録される.

イベント情報を 16 bit に詰め込み, それをサイズ 32 のリングバッファに記録する. タスク切替えに関するイベント情報は, 上位 10bit に時刻フィールド, 続く 1bit に種別フィールド, 残りの 5bit にタスク ID フィールドを持つ. 時刻フィールドには, タスク切替えが行われた時刻が 1μ 秒単位の精度で記録される. 種別フィールドには, タスク切替えが行われた理由が, タスクが終了またはスリープしたためか, 高優先度タスクによってプリエンプトされたためであるかが記録される.

また, 割込みハンドラの起動に関するイベント情報は, 上位 10bit に要因番号フィールドを持つ. このフィールドには, 割込みの要因番号が記録される. また, タスク切替えとのイベント情報を区別するために, イベント情報の下位 6bit(タスク切替えの種別フィールドとタスク ID フィールドに相当) に 0 を書き込んでいる.

3.2 実行時間の算出

ここでは, 前述した方法により取得したシステムの動作ログを用いて, タスクの実行時間を推定する方法を説明する.

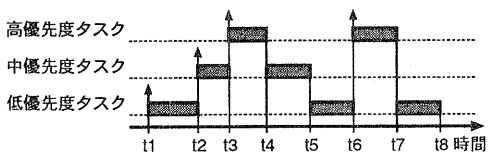

図 2: タスクの実行時間の算出

例として,3 つの異なる優先度をもつタスクが実行される様子を図 2に示しているが, 簡単のため, カーネルへの処理の移行を省略し, 割込みハンドラの実行はないものしている.

低優先度タスクと中優先度タスクは, より高優先度のタスクにプリエンプトされるため, 連続して実行されるだけでなく, 分割されて実行されることもある. 連続して実行された場合のタスクの実行時間は, 終了時刻と開始時刻の差により簡単に求めることができる. 一方, 分割された実行から 1 つのまとまった実行時間は, 前述したタスク切替えの種別を用いて求めることができる.

つまり, タスク切替えの種別によって, タスク切替えをタスクの終了またはスリープに伴うタスク切替えと, プリエンプトによるタスク切替えを区別することができるから, 前者のタスク切替えが出るまで, 分割された処理の実行時間をすべて足し合わせればよい.

例えば, 図 2の低優先度タスクは時刻 $t1$ に起動して,$t8$ で終了し, その実行時間 C_L は,$(t2-t1)+(t6-t5)+(t8-t7)$ であるということができる.

3.3 タスクのサブスケジューリング

エンジン制御システムのいくつかのタスクは, その内部でさらにスケジュールするものがある. つまり, タスク内で行われる処理がそれぞれ周期をもっており, タスクの起動ごとに処理される処理の内容が異なる. これをタスクのサブスケジューリングと呼ぶ.

当然ながら, 実行時間も周期的に変動するため, サブスケジューリングを考慮しないで求めた最大実行時間を用いた CPU 使用率やスケジュール可能性解析は実際よりも悲観的となる.

サブスケジューリングを考慮する場合, タスク内で処理される各処理の起動周期の最小公倍数を求めて, タスクの実行をその数でサブスケジュール単位に分解し, 各サブスケジュール単位ごとの最大実行時間を求める必要がある. サブスケジューリング単位は, 後述するマルチフレームタスクのフレームに相当する.

3.4 タスク切替えに伴うオーバヘッド

システムの時間的振舞いを正確に知るためには, タスクの切替えに伴うオーバヘッドを考慮する必要がある. タスクの切替えに伴うオーバヘッドは, 低優先度タスクがプリエンプトされてから高優先度タスクが実行されるまでの時間と, 高優先度タスクの実行が終了してから再び低優先度タスクに実行が移るまでの時間の合計である.

タスクの切替えに伴うオーバヘッドは, 高優先度タスクの実行時間に含めるべきであることが知られている. 低優先度タスクにタスク切替えに伴うオーバヘッド時間を含めると, その実行中に発生したプリエンプション回数に伴って, 低優先度タスクの実行時間が変動するが, 高優先度タスクに含めると, タスクの実行時間に含まれるオーバヘッドは一定 (タスク切替え 1 回分) として扱うことができるからである.

しかし本研究では, タスクの切替えが行われた時刻をディスパッチ処理の中程でタイマからの読み込みを行っているため, 高優先度タスクへのタスク切替えでのタイ

マの読み込み以前の処理に要する時間と, 低優先度タスクへのタスク切替えでのタイマの読み込み以降の処理に要する時間は, 低優先度タスクに含まれることになる.

この低優先度タスクに含まれる, 本来含まれるべきでないオーバヘッドの時間を推定し, 低優先度タスクからそれを減算した. タスク切替えに伴うオーバヘッドの時間を, あるタスクがプリエンプトされた回数とその時のの実行時間の関係から求めたが, 使用している CPU のクロック周波数やタスク切替え時の退避/復帰レジスタ数などから, 妥当な値であることが確認できている.

3.5 割込みハンドラの実行時間の推定

以上までの議論では, タスクの実行中に割込みハンドラが実行される場合を考慮できていない. システムの時間的振舞いをより正確に知るためには, 割込みハンドラの実行時間を推定する必要がある.

割込みハンドラの実行時間は, 割込みハンドラの終了時刻と起動時刻との差により求めることができる. しかし現在のところ, 記録装置の取得データ量に制限があるため, 割込みハンドラの関しては, その起動イベントしか記録できていない.

そこで, タスクの実行時間の分布から割込みハンドラの実行時間を推定する方法をとることにした. 基本的な考え方は, ある割込みハンドラが 1 回だけ実行された場合のタスクの実行時間と, どの割込みハンドラが全く実行されなかった場合のタスクの実行時間との差をとれば, その割込みハンドラの実行時間を求めることができる, というものである.

種類の異なる複数の割込みハンドラが実行された場合でも, 同様の方法を繰り返し行うことで求めることができるが, 推定の信頼性を高く保つために, 割込みハンドラが起動された回数が少ない場合を使用している.

この方法で求めた時間は, 割込みハンドラのみの実行時間ではなく, 割込みハンドラの出入口処理 (割込みハンドラの起動の前後に必ず行われる処理) も含まれている.

3.6 CPU 使用率による評価

本研究では, スケジュール可能性解析を行うための前段階として,CPU 使用率を用いた評価を行った. この評価を, 最大実行時間を用いた場合の評価と実際のシステムの挙動とのずれを見積もるとともに, サブスケジューリング, 非周期タスクおよび QoS 制御を考慮した場合の解析の精度を見積もる方法の一つであると位置付けている.

具体的には, 静的な優先度割付けで最適なアルゴリズムである Rate Monotonic (RM) 法を適用し, 前述した方法で算出したタスクの実行時間と割込みハンドラの実行時間を用いて, すべてのタスクと割込みハンドラの CPU 使用率の総和を求めた. CPU 使用率 U は, 次式で表される.

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i}$$

ここで,n は全タスク数,C_i と T_i はそれぞれ, 優先度が高い順に並べた場合の,i 番目のタスクの実行時間と起動周期である. CPU 使用率は, システムのスケジュール可能性を判断する場合の基本的な量となる. 単一プロセッサの場合, 最大で利用できる CPU 使用率は 1 であり, システム全体がスケジュール可能であるためには, まず CPU 使用率は 1 以下でなくてはならない.

また, サブスケジューリングを考慮する場合, m_i を, タスク i をサブスケジューリング分割した数 (サブスケジューリングをしないタスクは $m_i = 1$ である), その各サブスケジュール単位の最大実行時間をそれぞれ, $C_i^1, C_i^2, \cdots, C_i^{m_i}$ と書くことにすると, CPU 使用率は次のように書き換えることができる.

$$U = \sum_{i=1}^{n} \sum_{j=1}^{m_i} \frac{C_i^j}{m_i T_i} \qquad (1)$$

まず, 式 (1) の C_i^j に, サブスケジューリングを考慮しない場合と考慮した場合のタスクの最大実行時間を用いて, エンジン回転数に対するシステムの CPU 使用率を求めた. 図 3にそれぞれの CPU 使用率とアイドルタスクの動作から求めた実際の CPU 使用率を示している. 図 3

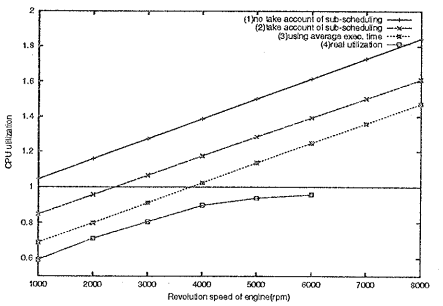

図 3: 回転数に対する CPU 使用率
(1) サブスケジューリングを考慮しない場合
(2) サブスケジューリングを考慮した場合
(3) タスクの平均実行時間を用いた場合
(4) 本当の CPU 使用率

によると, サブスケジューリングを考慮した場合はサブスケジューリングを考慮しない場合に比べ,CPU 使用率の評価の正確さがおよそ 20 %の向上されることが分かる. それでも, 回転数が 2000rpm あたりで CPU 使用率が 100 %を超えているので, これ以上の回転数ではシステム全体がスケジュール可能であると判定できない. しかし, 本研究で対象とした評価用の ECU は,4000rpm を

超えるまで正常に動作しているため, 求めた CPU 使用率が正確でないと言える[1].

参考として, 最大実行時間に代えて平均実行時間を用いた場合の CPU 使用率を算出し, 同図に示している.

3.7 非周期タスクと QoS 制御の取り扱い

前節の求めた CPU 使用率が正確ではない理由として, (1) 非周期タスクの取り扱いが適切でない, (2)QoS 制御が考慮されていないの 2 つが挙げられる.

他の制御システムとの通信を行うタスクは, 外部からの要求により非周期に起動され, どのような周期パターンで起動されるかが分からない. そこで今回は, 設計段階で分かっている最小起動周期を用いている. 動作ログから算出した平均起動周期は, 最小起動周期の 5 倍から 10 倍を超える範囲にある. 最小起動周期が解析結果を悲観的なものとしている.

参考として, 非周期タスクの起動周期に動作ログから算出した平均起動周期を用いた場合の CPU 使用率を図 4に示す. システム中の非周期タスクの動作がまだ明らかとなっていないために, 評価に設計上の最小起動周期を採用せざるをえないが, 起動周期に何らかのパターンを見出すことができれば, より正確な評価を行うことができると考えられる.

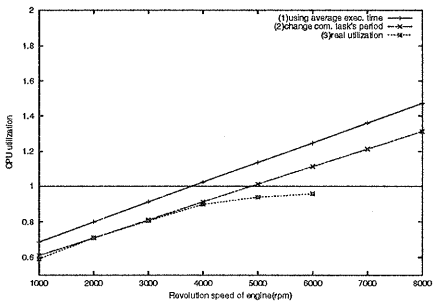

図 4: 非周期タスクの取り扱い
(1) 非周期タスクに平均起動周期を用いた場合
(2) 本当の CPU 使用率

エンジン制御では, 負荷 (CPU 使用率) はエンジン回転数の関数となり, 一般に高回転ほど高くなる. 実用域を超えた高負荷が原因で,CPU が止まる[2] 危険を防ぐ対策として, 低優先度タスクの実行を間引くようにしている.

実際に, エンジン回転数が 4000rpm 前後で, 優先度の低い方から 2 つのタスクの実行が間引かれ始め,6000rpm では最低優先度タスクは 2 割程度しか実行されない結果が得られている[3].

もし, 低優先度のタスクの実行を間引いても制御性が成り立つのであれば, モードチェンジ法をという手法を採用し,4000rpm 以上の高回転域でタスクの実行を間引くことを考慮することによって, 体系的に負荷を低減することも可能である. エンジンの回転数に対してどれくらい処理を間引いたらよいのかを制御の面から明らかにしていくことが, 今後の課題として残っている.

4 スケジュール可能性解析

一般化されたマルチフレームタスク (GMF タスク) からなるタスクセットのスケジュール可能性に関する研究が行われている. マルチフレームタスクモデル [1] は, タスクの実行時間があるパターンにしたがって大きく変動する状況を効率的に扱うための枠組みで, エンジン制御を行うタスクを有効にモデル化することができる.

本研究では, スケジュール可能性解析に GMF タスクのモデルを採用し,GMF タスクセットのスケジュール可能性の必要十分条件を効率的にチェックする方法として提案されている,MaximumInterference Function(MIF) [3, 4] の概念を用いた.

GMF タスクや MIF の詳しい説明は [3] を参考にされたい.

4.1 一般化されたマルチフレームタスク (GMF タスク)

以下に GMF タスクの定義を述べる. N_i 個のフレームを持つ GMF タスク T_i は,j 番目のフレームを 3 つ組 (C_i^j, D_i^j, P_i^j) で表すことにすると, 長さ N_i の 3 つ組の列 $((C_i^0, D_i^0, P_i^0), \cdots, (C_i^{N_i-1} D_i^{N_i-1} P_i^{N_i-1}))$ によって特徴づけられる. ここで,C_i^j は T_i の j 番目のフレームにおける最大実行時間,D_i^j は T_i の j 番目のフレームの相対デッドライン,P_i^j は j 番目と $j+1$ 番目のフレームの最小間隔を示す.

つまり,T_i の (通算して)k 番目のフレームの起動時刻を a^k,(絶対) デッドラインを $a^k + d^k$, 実行時間を c^k とすると, 次の関係が成立する.

$$a^0 \geq 0, a^{k+1} \geq a^k + P_i^{k \bmod N_i}$$
$$d^k = D_i^{k \bmod N_i}$$
$$c^k \leq C_i^{k \bmod N_i}$$

4.2 Maximum Interferance Function(MIF)

MIF は, 各タスク T_i が低優先度のタスクを邪魔する最大時間を経過時間の関数である. より厳密には, タスク T_i が, いずれかのフレームが起動されてから時間 t の間

[1]製品用の ECU では, 7000rpm を超える範囲まで正しく動作することが確認されている.

[2]最低優先度タスクが長い時間実行されない状況 (本来あり得ない) が続くと, マイコンがシステム異常と判断し,CPU へのリセット信号を発行する.

[3]今回対象とした評価用の ECU では実用域で処理が抜けてしまい, このような結果となっているが, 製品用の ECU では実用域で処理が抜けないため, これが問題にならないことに注意されたい.

に低優先度のタスクを邪魔する最大時間の関数が MIF であり, $M_i(t)$ と表現する. 詳しくは [3] を参照して頂きたい.

ここで,GMF タスクの MIF $M_i(t)$ を定義するために, まず,E_i^k を以下のように定義する. タスク T_i の k 番目のフレームが起動されてから, 時間 $t(>0)$ が経過するまでの間に,T_i が起動する処理の実行時間の合計の最大値を $E_i^k(t)$ と書く.

$$E_i^k(t) = \sum_{h=k}^{k+J-1} C_i^{h \bmod N_i} + \min\left(C_i^{(k+J) \bmod N_i}, t - \sum_{h=k}^{k+J-1} P_i^{h \bmod N_i}\right)$$

ここで,J は次の式を満たす最大の整数である.

$$\sum_{h=k}^{h+J-1} P_i^{k \bmod N_i} \geq t$$

これを用いて,GMF タスクの MIF $M_i(t)$ を定義すると次のように記述することができる.

$$M_i(t) = \max_{0 \leq k \leq N_i - 1} E_i^k(t)$$

4.2.1 2 つの MIF の加算

タスクセット全体がスケジュール可能であるためには. それに含まれる全てのタスクがスケジュール可能でなければならない. つまり, 最高優先度タスクから最低優先度タスクまで, 順に各タスクのスケジュール可能性を判定する必要がある. 今, 高優先度の GMF タスク T_1 から T_{i-1} までスケジュール可能であると判定されたとする. この時,GMF タスク T_i にとっては, どのタスクにどれだけ邪魔されたかではなく, 全ての高優先度タスクにどれだけ邪魔されたかを考えればよい. 全ての高優先度タスクにどれだけ邪魔されたかは, GMF タスク $T_1, \cdots, T_{i-1}$ の MIF $M_1(t), \cdots, M_{i-1}(t)$ の加算により知ることができる.

しかし, 通常の算術加算で得られた MIF は, その定義に反しているため, MIF のための加算を新たに定義する必要がある. 2 つのタスク T_1, T_2 の MIF $M_1(t), M_2(t)$ の加算は次のように定義できる.

$$M_1(t) \oplus M_2(t) = t - \max_{0 \leq \tau < t} [\tau - \{M_1(\tau) + M_2(\tau)\}]$$

上式の右辺第 2 項は, 時刻 t までにタスク T_1, T_2 のどちらも実行されなかった時間の総和である.

T_i の実行は, より高優先度をもつタスク $T_1, \cdots, T_{i-1}$ により邪魔されるから, それらのタスクの MIF $M_1(t)$, $\cdots$, $M_{i-1}(t)$ の加算を $M_{1,i-1}$ と表現すると,$M_{1,i-1}$ は次式のように記述できる.

$$\begin{aligned} M_{1,i-1}(t) &= M_1(t) \oplus \cdots \oplus M_{i-1}(t) \\ &= t - \max_{0 \leq \tau < t} \left[\tau - \sum_{k=1}^{i-1} M_k(\tau)\right] \end{aligned}$$

4.3 相対的なデッドライン

タスクの (相対的な) デッドラインが周期と比較して等しいか短い場合, クリティカルインスタントに起動されたタスクの最大応答時間を求めることで, 容易にスケジュール可能性を評価することができる. つまり, 最大応答時間が相対デッドラインと等しいか短ければスケジュール可能であり, そうでなければスケジュール不可能である.

一方, タスクの (相対的な) デッドラインが周期より長い場合は, 同じタスクが複数, 同時に実行可能となる可能性があるため, 簡単にはスケジュール可能性を判定することはできない. 前に起動されたタスクが, 起動周期よりも長い応答時間を持つ場合, 次に起動される同じタスクの実行を妨害するため, 次に起動されるタスクの応答時間に, 前に起動されたタスクが次の起動時刻からはみ出して実行する時間を考慮する必要がある.

そこで,busy period の概念を用いる [5]. busy period とは, 簡単には, ある優先度よりも高いかまたは等しい優先度をもつすべてのタスクが同時に起動された時刻 t_0(t_0 でどのタスクも実行されていないという条件もつく) から, そのすべてが終了する時刻 t までの区間 $(t_0, t]$ のことである.

busy period を求めるには, 簡単には, 今スケジュール可能性解析の対象となるタスクの優先度よりも高いかまたは等しい優先度をもつ, すべてのタスクの MIF を足し合わせて,MIF の傾きが初めて 0 となる (グラフで平坦になる) 時刻を t とすればよい.

busy period が, 対象となるタスクの優先度よりも高いかまたは等しい優先度をもつ, すべてのタスクの起動周期の最小公倍数よりも大きい場合, 明らかにスケジュール不可能で最大応答時間も無限大となるので, これ以上の解析を取り止める.

そうでない場合は,busy period の中に対象となるタスクが何回起動されるかを求め, それぞれに対応する応答時間を計算し, 最大のものを最大応答時間として決定する. そして, 最大応答時間が相対デッドラインより小さい場合はスケジュール可能であり, そうでない場合はスケジュール不可能であると判定できる.

相対的なデッドラインが周期より長いタスクがサブスケジューリングを行っている場合の最大応答時間を求める議論は, 行っていない場合よりも多少複雑になる. しかし, この場合も, まず busy period の求めることは変わらない.

例として, サブスケジューリング単位が 4 つある場合を考えると, あるサブスケジューリング単位から起動を

始めた場合を考えて,(仮の) 最大応答時間を求める. この時点に,(最大で)4 つの最大応答時間が求まる. これをすべてのサブスケジューリング単位から起動を始めた場合で繰り返す. 各サブスケジューリング単位で,(最大で)4 つの最大応答時間の候補が挙がり, その中で最大のものを (真の) 最大応答時間と決定する.

付録に, busy period の概念を用いて任意の相対デッドラインをもつタスクの最悪応答時間を求めることができることの証明を示しているので, 興味のある方はこちらも参照して頂きたい. MIF を用いたスケジュール可能性解析の厳密な正当性の証明は今後の課題として残っている.

4.4 スケジュール可能性解析

システム動作の解析で求めた最大実行時間と, 前項で述べた方法を用いて, スケジュール可能性の評価を行った.

スケジュール可能性解析の具体的な例として, 今回対象としているシステムの中にあるタスク (起動周期と相対デッドラインは 4 ミリ秒) を挙げる. このタスク高優先度のタスクの MIF を足し合わせたものと, さらに 4 ミリ秒のタスクの MIF を加えたもの (5000rpm 時) を図に示している.

この場合,busy period の幅は約 2700μ 秒であるから, このタスクの最大応答時間も約 2700μ 秒となり, 相対デッドライン (4 ミリ秒) 以下であるから, このタスクはスケジュール可能である. このように, すべてのタスク

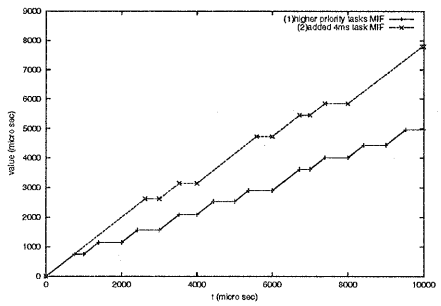

図 5: MIF を用いたスケジュール可能性解析の例
(1) 高優先度タスクの MIF を足し合わせたもの
(2) さらに 4 ミリ秒タスクの MIF を足し合わせたもの

についてスケジュール可能性の評価を行った結果, エンジン回転数が 2000rpm の場合は最低優先度タスクから 1 つ, 3000rpm から 5000rpm の場合で 2 つ,6000rpm の場合で 3 つがデッドラインを満たせないという結果が得られた. しかし, クリティカルなタスクのリアルタイム性は十分に保証されうるという結果が得られた.

5 まとめと今後の課題

本研究では, エンジン制御システムの設計に RMA の手法を適用するために, 実車で得られた動作ログを用いて, エンジン制御システムを構成するタスクの時間的振舞いを解析し, 各タスクの最大実行時間と割込みハンドラの実行時間を求めた. また, 得られたデータを用いて,CPU 使用率と必要十分条件を用いてスケジュール可能性の解析を行った. スケジュール可能性の解析では, タスクモデルに GMF タスクモデルを適用し, MIF と busy period の概念を用いた.

本稿で紹介した解析手法を用いることで, エンジン制御システムの時間的な振舞いを, その設計段階で (つまり, 車両に載せてテストを行うことなしに) 体系的な手法を用いて予測することが可能となり, 最大実行時間の評価によるクリティカルな処理のリアルタイム性を保証できる[4]ことが裏付けることができると考えられる.

本研究で用いた動作ログの取得・記録環境では, 今以上のシステムの動作に関する情報を得ることは難しいが, 今後の課題として挙げられる項目を検討することで, 将来的には, エンジン制御システムのより体系的な設計・開発手法を確立できると考えている.

今後の課題として, 前述した非周期タスクと QoS 制御の取り扱いの他に同優先度タスクの取り扱いとタスク間の依存関係が挙げられる.

今までのリアルタイムスケジューリングに関する研究は, タスクセット中の全てのタスクは, それぞれ異なる優先度を持つという仮定で行われている場合がほとんどである. この仮定がなされるのは, 同じ優先度を持つタスクが複数存在する場合の議論が複雑となるからである. しかし, 現実のシステムでは同じ優先度を持つタスクが複数存在する場合があり, 本稿で対象とするエンジン制御システムも, このケースに当てはまる. MIF を用いることで, 同優先度タスクの存在を比較的容易に取り扱うことができると予想している. つまり, 同優先度タスクには 1 度しか邪魔されないので,MIF の値がそれ以降増えない時刻が存在すると予測しているが, 厳密な検証は今後の課題として挙げられる.

各処理の依存関係には, 実行順序に関するものとリソース共有に関するものがある. タスクや割込みハンドラの実行順序に排他的な条件がいくつか存在するが, これらを考慮していないために, 起こり得ない条件が今回の評価結果に含まれており, 結果として悲観的な結果となっている可能性もある. また, 今回対象としているシステムは, セマフォなどの排他制御によりリソース共有を実現しているが, これらを考慮することも重要であると考えている. 今後, タスク間の依存関係を明確にすることで, より精度の高い解析が行えると考えられる.

[4]最大実行時間を実測で求めるため, 保証の限界はある

参考文献

[1] A. K. Mok and D. Chen, "A multiframe model for real-time tasks," in *Proc. Real-Time Systems Symposium*, pp. 22–29, Dec. 1996.

[2] S. Sasaki, "Proposal for on-chip debug resources for high-speed microcontrollers," in *Proceedings of the Embedded Systems Conference East 1997*, Miller Freeman, Mar. 1997.

[3] 高田広章, 坂村健, "マルチフレームタスクセットの静的優先度スケジューリングによるスケジュール可能性," 信学技報 *(1997年実時間処理に関するワークショップ RTP'97)*, vol. 96, no. 596, pp. 29–35, 電子情報通信学会, Mar. 1997.

[4] H. Takada and K. Sakamura, "Schedulability of generalized multiframe task sets under static priority assignment," in *Proc. Real-Time Computing Systems and Applications*, pp. 80–86, Oct. 1997.

[5] John P. Lehoczky, "Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines", in *Proc. Real-Time Systems Symposium*, pp. 201–209, Dec. 1990

[6] 菅沼 英明, 佐藤 浩司, "自動車における組込みシステム開発の現状－エンジン制御を中心として－," 情報処理, vol. 38, pp. 892–897, Oct. 1997.

[7] H. Suganuma,K. Sato and H. Takada, "A Study of Evaluating the Real-Time Property for Engine Control Software," SAE 2001 World Congress, Mar. 2001.

[8] 小濱 一師, 高田 広章, 菅沼 英明, 佐藤 浩司 "エンジン制御システムの解析手法に関する研究," 信学技報 *(1999年実時間処理に関するワークショップ RTP'99)*, vol. 98, no. 687, pp. 9–16, 電子情報通信学会, Mar. 1999.

[9] N. C. Audsley, A. Burns, R. I. Davis, K. W. Tindell, and A. J. Wellings, "Fixed priority preemptive scheduling: An historical perspective," *Real-Time Systems*, vol. 8, pp. 173–198, 1995.

[10] Giorgio C.Buttazzo, "HARD REAL-TIME COMPUTING SYSTEMS: Preditable Scheduling Algorithms and Applications," Kluwer Academic Publishers, 1997

付録

文献 [4] において, プリエンプティブな静的優先度スケジューリングのもとでの Frame Separation property(FS 性) を持つ GMF タスクセットに対する critical instant 定理が述べられている.

実はこれは,FS 性を持たない場合も同様に書け, 文献 [4] で述べられている Critical Instant Candidates の定義を用いれば次のように証明することができる.

定義 1 (Critical Instant Candidates) ある *GMF* タスクに対する *critical instant candidates* とは, そのタスクのあるフレームがすべてのより優先度の高いタスクのいずれかのフレームと同時に起動した状況で, 以降続くフレームを最小間隔で起動し, いずれのフレームにおいても最大実行時間を使い切る場合をいう.

定理 1 (Critical Instant 定理) ある *GMF* タスクのあるフレームに対する *critical instant* は, そのタスクに対する *critical instant candidates* のいずれかである.

証明. タスク T_i のあるフレームが時刻 t で到着したとき, それ以降のあるフレーム F が最悪応答時間を持ち, 時刻 t_{end} に終了すると仮定する. t と t_{end} のあいだは常に T_i 以上の優先度を持つタスクが実行されているものとする.t 以前で, より高い優先度を持つタスクが実行されていないようなもっとも遅い時間を t_0 とする. 時刻 0 ではどのタスクも実行されていないので,t_0 は必ず存在する.F の到着時刻を t_0 に変更したとしても, t_0 と t の間は少なくとも *1* つのより高い優先度を持つタスクが実行されているため, 依然として t_{end} に終了する.

それぞれのより高い優先度を持つタスクの t_0 後の最初の到着時刻が t_0 に移動したとする. この場合,F の終了時刻は $t_{end'}$ に遅延するかもしれない. それらの後に続くフレームの到着時刻が移動されるためにそれぞれのフレームはその最小間隔で要求され, それぞれのフレームがその最大実行時間を消費すると仮定する. これは F の終了時刻を $t_{end''}$ にさらに遅延させる. 今,t_0 は T_i の *critical instant candidates* の *1* つであり,F は最初の状況と同じかより長い応答時間を持つ.

F が最初の状況で最悪応答時間を持つという仮定から,F がその *critical instant candidates* に要求されたときも最悪応答時間を持つ. □

これを, 任意のデッドラインを持つ周期タスクモデルにおいて優先度 i のタスク T_i の最悪応答時間を求めるために Lehoczky によって提案された level-i busy period[5] の概念を用いれば,「タスク T_i のフレーム F の最悪応答時間は,T_i のいずれかのフレームが起動されると同時に, そのタスクより優先度の高いタスクがすべて同時にいずれかのフレームの実行を起動した瞬間から始まる level-i busy period のなかにある.」と言いかえることができる.