Vol.2014-SLDM-165 No.3
Vol.2014-EMB-32 No.3
2014/3/15

高応答性を要求するリアルタイムアプリケーション統合のためのARINC653拡張スケジューリングアルゴリズム

佐藤 祐一[1]　松原 豊[1]　本田 晋也[1]　高田 広章[1]

概要：車載制御システムやロボット制御システムには，エンジンやモータ等を制御するために，高い応答性を要求するリアルタイムアプリケーションが搭載されている．本論文では，複数のリアルタイムアプリケーションを単一のプロセッサ上に統合するためのスケジューリングアルゴリズムを提案する．航空機分野のソフトウェアプラットフォームの標準規格である ARINC 653 をベースに，各アプリケーションの独立性を維持しつつ，高い応答性を要求する処理を特権割込み処理という特別な割込み処理によって実現できるよう拡張したアルゴリズムである．実際の車載制御アプリケーションに対して適用した結果，各アプリケーションの時間制約を満たしつつ，それらの実行周期を ARINC 653 の約 3 倍，周期リソースモデルの約 5.9 倍に延ばすことができることを確認した．この結果から，従来手法に比べて，現実の環境に適用し易いアルゴリズムであることを述べる．

キーワード：リアルタイムシステム，階層型スケジューリングアルゴリズム，ARINC 653

1. はじめに

代表的な分散リアルタイムシステムの 1 つである自動車制御システムには，走行性能の向上や，運転支援を目的とした機能が数多く搭載されている．それにともない，各機能を実現するための制御コンピュータである ECU（Electronic Control Units；電子制御ユニット）の数が増加している．ロボット制御システムにおいても，多数のモータが搭載され，それらを制御するコンピュータが数多く搭載されている．

このような制御システムに対して新しい制御機能を追加する場合，新しいコンピュータを追加する方法と，既存のコンピュータに制御アプリケーションを追加する方法が考えられる．車載制御システムの開発では，これまで前者の方法が採られ，ECU 搭載数の増加によるハードウェアコストの増加や，ECU 設置のためのスペース不足といった問題が発生している．後者の方法では，あるアプリケーションで発生した障害が他のアプリケーションに影響を与える可能性ある．例えば，あるアプリケーションが想定以上のプロセッサ時間を使ったことで，別のアプリケーションが時間制約を満たせなくなることや，安全に関係ないアプリケーションの不具合により，安全関連のアプリケーションが正常に動作しなくなってしまうといった問題が発生する．

本論文では，個別に開発・検証された複数のリアルタイプアプリケーションを単一のプロセッサ上で統合して動作させること（アプリケーション統合）を想定し，アプリケーションごとのプロセッサ時間とメモリ領域を保護するスケジューリングアルゴリズム（パーティショニング）を検討する．パーティショニングは，航空機向けのソフトウェアプラットフォームで標準的に採用されている [1]．さらに，自動車制御システムを対象とした機能安全規格 ISO 26262 にも盛り込まれるなど，様々な分野で注目が高まっている [2]．

本論文では，高応答性を要求する処理が存在するアプリケーションを統合する場合でも，アプリケーションの時間制約を満たすことができ，かつリソースが限られるシステムでも実現できるよう，アプリケーションの切替え頻度を抑えることができるバランスの良いスケジューリングアルゴリズムを提案する．実際の車載アプリケーションに対して適用し，アプリケーションの時間制約を満たすための設計パラメータを計算する．従来手法と比較評価した結果，提案アルゴリズムでは，各アプリケーションの時間制約を満たしつつ，もっとも長い実行周期を選択できることを確認する．

本論文の構成を述べる．まず，第 2 章で関連研究について述べ，主に ARINC 653 の問題点を指摘する．第 3 章で，

[1] 名古屋大学 大学院情報科学研究科
Graduate School of Information Science, Nagoya University

ⓒ 2014 Information Processing Society of Japan

提案アルゴリズムについて，動作例を交えて詳細に説明する．第 4 章では，提案アルゴリズムを評価するために，2 つの車載アプリケーションを対象に，アプリケーションの時間制約を満たす設計パラメータを計算し，従来手法と比較する．最後に，第 5 章で結論と今後の課題を述べる．

2. 関連研究

パーティショニングのためのスケジューリングアルゴリズムとして，アプリケーションをスケジューリングするグローバルスケジューラと，アプリケーション内のタスクをスケジューリングするローカルスケジューラを階層的に配置した階層型スケジューラが数多く提案されている．

もっとも厳密なパーティショニングを実現するアルゴリズムの 1 つに，米国 ARINC（Aeronautical Radios, Inc.）社が発行する，航空機の統合制御コンピュータである IMA（Integrated Modular Avionics）向けの基盤ソフトウェア標準規格 ARINC 653 がある [4]．このアルゴリズムでは，設計段階で，各アプリケーションの実行順序と，すべてのアプリケーションを 1 回以上実行する一連の実行周期（システム周期と呼ぶ）に対する各アプリケーションの実行時間の割合（プロセッサ帯域）を決定する．実行時には，これらの設計パラメータに基いて，システム周期として設定した周期で，すべてのアプリケーションを順番に実行することを繰り返す．これにより，アプリケーションごとのプロセッサ帯域を厳密に保護することができる．しかしながら，仕様上，割込み処理の考え方がないので，高い応答性を要求するアプリケーションには向かない．

例えば，表 1 に示すタスクセットを ARINC 653 でスケジュールすることを考える．アプリケーション A，B の順に実行すると想定すると，図 1 のようにスケジュールされる．このとき，時刻 15 で起動した τ_2^A のように，自身が所属するアプリケーションの実行時間が終了した直後に起動すると，次の周期まで実行が待たされてしまう．自動車制御システムやロボット制御システムのように，高応答性を要求する処理が存在するアプリケーションを統合する場合には，アプリケーションの実行頻度を高めるために，他のすべてのアプリケーションも含めて短いシステム周期で実行を切替えるか，必要以上のプロセッサ帯域を割当てて当該アプリケーションの実行時間を延ばす必要がある．したがって，このアルゴリズムをリソースが限られる組込みシステムで実現することは，現実的には難しい場合が多い．

それに対して，統合前に時間制約を満たすアプリケーションが統合後でも時間制約を満たすことを保証できるアルゴリズムが提案されている [5], [6]．これらのアルゴリズムでは，デッドラインが迫っている処理からスケジューリングすることで，理論的には，スケジュール可能な最低限のプロセッサ帯域を割り当てれば良いという特徴である．しかしながら，高応答性を要求する処理が存在するアプリケーションを統合する場合には，アプリケーションを切り替える頻度が高くなってしまうことが分かっている [7]．また，これらのアルゴリズムは，動作が複雑であるため，パーティショングできていることを第 3 者に説明するのが難しいという問題がある．

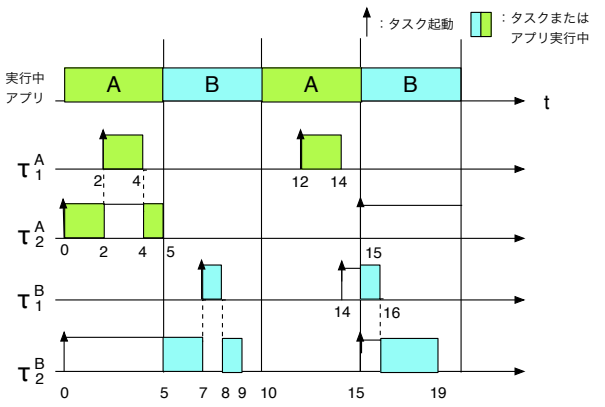


図 1 ARINC 653 によるスケジューリングの例

アプリケーションが時間制約を満たすことを保証でき，かつアルゴリズムの動作が単純なアルゴリズムとして，周期リソースモデル（PRM と呼ぶ）がある [8]．このアルゴリズムでは，アプリケーションごとに，その時間制約を満たせるよう実行周期とプロセッサ帯域を設計段階で計算しておき，実行時にはそれらに従ってスケジューリングするだけで良い．統合するアプリケーションが変わっても，設計パラメータを再計算する必要がないという点で独立性が高く，実行時の動作も単純であるという特徴がある．しかしながら，各アプリケーションに割当てるプロセッサ帯域は，本来アプリケーションが必要とする最低限のプロセッサ帯域よりも，余分に割当てる必要がある．また，現実のシステムで適用した場合に発生する，スケジュールによる処理オーバヘッドを評価した研究も行われている [10], [11], [12].

3. ARINC 653 拡張スケジューリングアルゴリズム

3.1 提案アルゴリズムの概要

第 2 章で述べたように，ARINC 653 には，高い応答性を要求される処理の実行が待たされることで，スケジュール可能性が下がるという問題がある．提案するアルゴリズムでは，設計段階で，特に高い応答性を要求する処理を特別な割込み処理（特権割込み処理と呼ぶ）として指定しておき，実行時に，処理要求が発生した時点で即座に実行することで応答性を向上する．

図 1 に示すように，特権割込みを実行することで，実行が後回しになったアプリケーションは，特権割込み処理が完了した後に実行を再開される．その実行時間は，特権割込み処理が動作した時間分だけ延長される．このように提

表 1　アプリケーションに所属するタスクのパラメータ

タスク	優先度	起動周期	実行時間	相対デッドライン	初期起動時刻
τ_1^A	1	10	2	10	2
τ_2^A	2	15	3	15	0
τ_1^B	1	7	1	7	7
τ_2^B	2	15	3	15	0

案アルゴリズムでは，特権割込み処理を優先的に実行することによって，アプリケーションの実行開始時刻と終了時刻が変動する可能性がある．しかし，システム周期に対する，各アプリケーションの実行時間の割合（プロセッサ帯域）を保証するために，システム周期の最後に，特権割込み処理を実行する分だけ余裕区間（アイドルウィンドウ）を設ける．あるシステム周期において特権割込み処理が実行されても，システム周期が変動しないよう，その実行時間分だけアイドルウィンドウが短くなる．アイドルウィンドウに対して設定するプロセッサ帯域は，システム周期の 1 周期内において発生しうる特権割込み処理の最大実行時間の総和以上を割当てるものとする．

3.2　アイドルウインドウに割当てるプロセッサ帯域

本節では，アイドルウィンドウに割当てるプロセッサ帯域の計算方法について述べる．前節で述べたように，アイドルウィンドウに対して設定するプロセッサ帯域は，システム周期の 1 周期内において発生しうる，特権割込み処理の最大実行時間の総和以上を割当てるものとする．

ここで，システム周期を T_{system}，特権割込み処理に指定した処理の集合を SP，特権割込み処理 τ_i の最大実行時間を e_i，最小到着間隔を T_i とする．まず，アイドルウィンドウの時間を IW は，次のように計算できる．

$$IW = \sum_{\tau_i \in SP} e_i \lceil \frac{T_{system}}{T_i} \rceil$$

さらに，アイドルウインドウに設定すべき最低限のプロセッサ帯域 U_{SP} は，次の式で計算できる．

$$U_{SP} = \frac{IW}{T_{system}}$$

3.3　動作例

提案アルゴリズムを用いて，２つのアプリケーションを統合した際のスケジューリング例を説明する．統合するアプリケーションは A と B の２種類で，それぞれのアプリケーションには，表 1 のタスクが存在する．各アプリケーション内では，固定優先度ベースのスケジューリングアルゴリズムによってタスクをスケジューリングするものとする．

ここでは，表 1 の τ_1^B を特権割込み処理として指定したとする．スケジューリングテーブル 1 周の内で，τ_1^B は，最

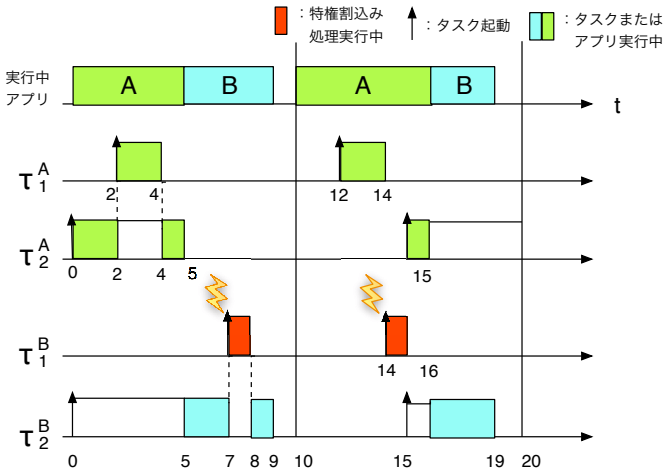


図 2　提案アルゴリズムによるスケジューリングの例

大２回起動するので，アイドルウィンドウは２単位時間以上設ける必要がある．提案アルゴリズムでスケジュールした例を図 2 に示す．時刻 14 で，τ_1^B は特権割込み処理であるため，アプリケーション A の実行に優先して実行される．１周目の区間（時刻 1 から時刻 10 まで）と，２周目の区間（時刻 10 から時刻 20 まで）のそれぞれで，特権割込み処理である τ_1^B は 1 回ずつ起動した．この場合，アイドルウインドウを 1 単位時間とすることで，システム周期を一定に保つことができる．

4.　車載アプリケーションへの適用

4.1　統合対象のアプリケーション

提案アルゴリズムを実際の車載制御アプリケーションに適用した事例を述べる．今回用いたのは，モータジェネレータのアプリケーション（MG アプリケーションと呼ぶ）と，エンジンの制御アプリケーション（ENG アプリケーションと呼ぶ）の２つである．これらのアプリケーションでは，制御対象のエンジンやモータの回転数が高くなると，タスクや割込み処理の起動間隔が短くなり，要求される応答時間も短くなる傾向にある．特に，MG アプリケーションでは，高い応答性を要求される処理が割込み処理として実装されており，短い相対デッドラインが設定される．

ENG アプリケーションは，39 個のタスクと 30 個の割込み処理で構成される．タスクと割込み処理は，時刻やエンジン回転に同期して周期的に起動するものと，非周期に起動するものがある．また，中には，起動する度に実行時

ⓒ 2014 Information Processing Society of Japan

間が変動するものがある．今回は，一般化されたマルチフレームタスクモデル（GMF）を用いて，アプリケーションが要求するプロセッサ時間を計算した [3]．

MG アプリケーションは，1 個のタスクと 9 個の割込み処理で構成される．このアプリケーションでは，おおよそ，制御対象のモータの回転数が高くなると，タスクや割込み処理の起動間隔が短くなり，要求される応答時間も短くなる．高い応答性を要求される処理のスケジューリングおいては，相対デッドラインを短くすることで実現する．今回は，最も負荷の高くなる 10,000rpm で回転する際の制御動作を対象とした．

各アプリケーションが統合前に動作していたプロセッサが持つ動作周波数（ENG と MG は，それぞれ 128MHz と 192MHz で動作していたと想定）を参考に，仮想的な統合プロセッサの動作周波数を 384MHz と想定する．各タスクの統合後の実行時間は，統合前に比較して，それぞれ 3 分の 1 と，2 分の 1 に短縮されるものとした．これらのアプリケーションは，統合前に，固定優先度ベーススケジューリングでスケジュール可能である．本論文では，統合後も，各アプリケーション内では，固定優先度ベースのスケジューリングアルゴリズムによってタスクをスケジューリングするものとする．

4.2 アプリケーションに割当てるプロセッサ帯域の計算

各アプリケーションに割当てるプロセッサ帯域は，ARINC 653 向けのパラメータを計算する手法を用いて計算する [9]．まず，特権割込み処理を指定せずに，すべてのタスクと割込み処理を ARINC 653 でスケジュールする場合に，各アプリケーションに割当てるべきプロセッサ帯域を計算した．その後，MG アプリケーションの相対デッドラインの短い処理から順に 1 つずつ，特権割込み処理として指定した場合に必要なプロセッサ帯域の変化を調べた．MG アプリケーションに要求される応答性は，ENG アプリケーションに比較して非常に短い（すなわち，設定される相対デッドラインが短い）ので，本論文では，MG アプリケーションの一部の割込み処理だけを特権割込み処理として指定する対象とした．

ARINC 653，PRM，提案手法の 3 つのアルゴリズムを用いて，アプリケーションごとにスケジュール可能となる，リソース供給周期とプロセッサ帯域を計算した結果を図 3 に示す．リソース供給周期は，対象アプリケーションに対してリソースを供給する周期を示しており，本論文では，アプリケーションの実行周期を意味する．なお，この図では，提案手法を適用した場合に確保すべきアイドルウインドウ用のプロセッサ帯域については表記していない．これを踏まえたスケジュール可能性の判断については，次節で述べる．

まず，ARINC 653 と PRM の結果を比較する．ENG アプリケーションについては，ARINC 653 と PRM を適用した場合，リソース供給周期が 70μs を越えると，ARINC 653 を適用する方が，必要なプロセッサ帯域が少なくて済むという結果が出ている．これは，ENG アプリケーションが実行を待たされる最大時間が，ARINC 653 を適用した方が，PRM を適用するよりも短くて済むからである．MG アプリケーションについても同様で，同じリソース供給周期であれば，ARINC 653 を適用する方が，PRM を適用する場合に比べて少ないプロセッサ帯域で済む．例えば，リソース供給周期を 60μs とする場合，ARINC 653 では 78%程度のプロセッサ帯域を割り当てれば良いが，PRM で 90%程度のプロセッサ帯域を割当てる必要がある．

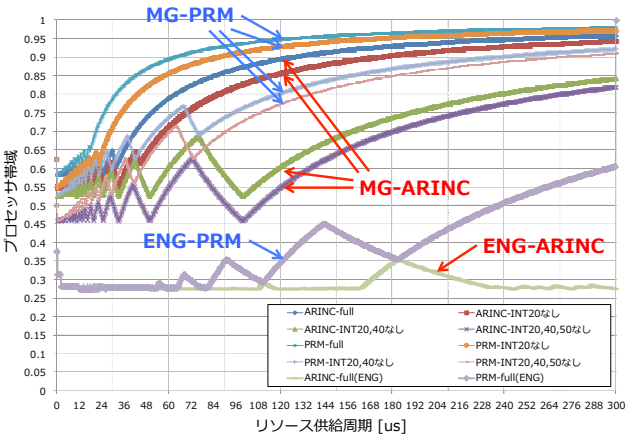


図 3 ARINC 653，PRM，提案アルゴリズムにおけるパラメータ計算結果の比較

次に，ARINC 653 と提案アルゴリズムの結果を比較する．ARINC 653 ではアプリケーションに含まれるすべての処理をパーティション内で実行するので，必要なプロセッサ帯域が最も多くなる．最も高い応答性を要求する（すなわち，相対デッドラインが最も短い）割込み処理である INT20 から順番に，INT40，INT50 を特権割込み処理として追加指定すると，パーティション内で実行すべき処理が減るので，段階的に必要なプロセッサ帯域も減る．必要なプロセッサ帯域の減少量は，特権割込み処理に指定する処理の負荷と，リソース供給周期に依存するが，例えば，MG アプリケーションに提案アルゴリズムを適用し，リソース供給周期を 120μs とした場合，INT20 を特権割込みとして指定した時点で 5%（90%から 85%），INT40 を追加すると 30%（90%から 60%），さらに INT50 を追加すると 35%（90%から 55%）減少した．

4.3 スケジュール可能性を保証する設計パラメータの計算

前節で述べたように，提案アルゴリズムでは，特権割込み処理に指定した処理が使用するプロセッサ帯域として，アイドルウインドウを設定する必要がある．提案アルゴリズムの適用条件としては，アプリケーションに割当てるプ

ⓒ 2014 Information Processing Society of Japan

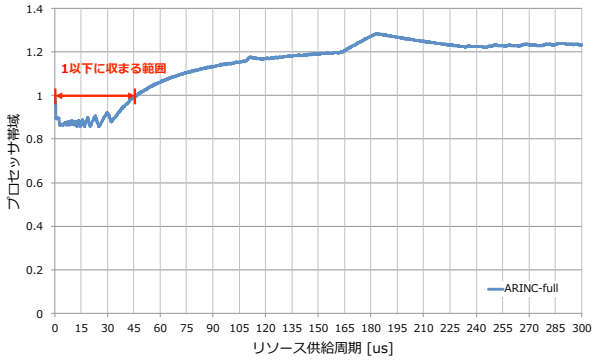


図 4 ARINC 653 におけるパラメータ計算結果

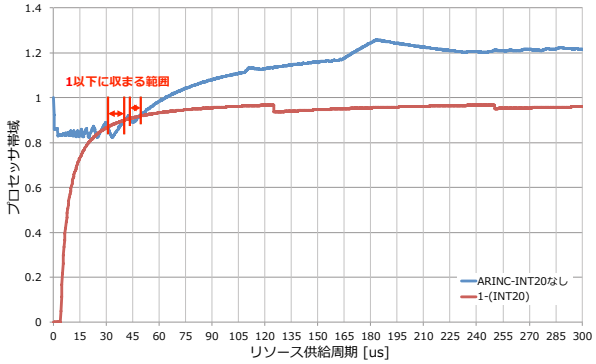


図 5 提案アルゴリズムにおけるパラメータ計算結果（INT20 を特権割込みとして指定）

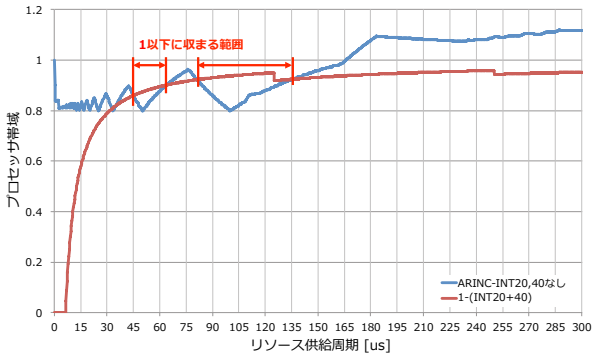


図 6 提案アルゴリズムにおけるパラメータ計算結果（INT20 と INT40 を特権割込みとして指定）

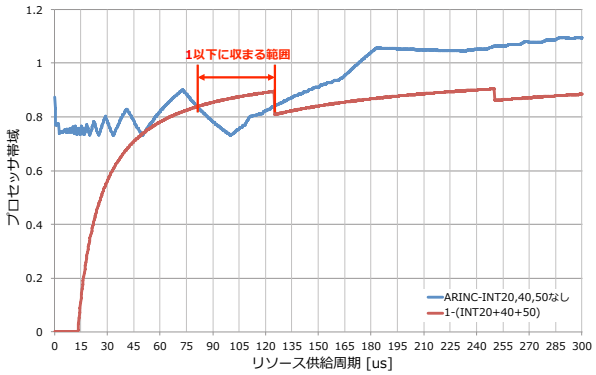


図 7 提案アルゴリズムにおけるパラメータ計算結果（INT20, INT40，INT50 を特権割込みとして指定）

ロセッサ帯域と，アイドルウインドウ用のプロセッサ帯域の合計が 1 以下である必要がある．スケジューリングの処理オーバヘッドを出来る限り小さくするためには，プロセッサ帯域の合計が 1 以下となる中で，最大のリソース供給周期を選択することが望ましい．本節では，プロセッサ帯域の合計に着目して，アルゴリズムを比較する．なお，PRM においても，プロセッサ帯域の合計が 1 以下となることを適用可能条件とするため，アプリケーションをスケジューリングするアルゴリズムは EDF（Earliest Deadline First）を想定する．

図 4〜図 7 に，MG アプリケーションに割当てるべきプロセッサ帯域を計算した結果を示す．図 4 は，特権割込み処理を指定せずに，すべてのタスク，割込み処理を ARINC 653 でスケジュールする場合のプロセッサ帯域である．リソース供給周期が，$[0,45]\mu s$ の場合に限り，プロセッサ帯域が 1 以下となるので，シングルプロセッサでスケジュール可能性であることを意味する．

相対デッドラインの短い割込み処理 3 つを特権割込み処理として指定する．相対デッドラインが短い順に，INT20, INT40，INT50 とする．特権割込み処理として，INT20 のみを指定した結果を図 5 に，INT20 と INT40 を指定した結果を図 6 に，INT20，INT40，INT50 を指定した結果を図 7 に示す．第 2 章で述べたように，特権割込み処理を指定した場合には，この特権割込み処理を優先的に実行した結果，実行中のアプリケーションの実行が遅延する可能性がある．そこで，システム周期を一定に保つためのアイドルウインドウを設定する．図 5〜図 7 では，アイドルウインドウに割当てるべきプロセッサ帯域を 3.2 節で述べた方法で計算し，1 から差し引いたプロセッサ帯域（すなわち，アプリケーションで使用可能なプロセッサ帯域の上限）も示している．したがって，図 5〜図 7 においては，アプリケーションとアイドルウインドウに割当てるプロセッサ帯域の合計が 1 以下となる範囲で，リソース供給周期を選択する必要がある．図 5〜図 7 から明らかなように，INT20 と INT40 の 2 つを特権割込み処理として指定した場合に，もっともリソース供給周期を延ばすことができることが分かった．

図 8 に，PRM のパラメータ設計結果を示す．リソース供給周期が $[0, 23]\mu s$ でのみ適用であることが分かる．これは，図 4 で示した ARINC 653 への適用結果に比べると，短いリソース供給周期を設定しなければならないことを意味する．言い換えると，同じリソース供給周期であれば，ARINC 653 の方が PRM よりも少ないプロセッサ帯域で済む．提案アルゴリズムは，ARINC 653 に対して，特権割込み処理の仕組みを導入して拡張したアルゴリズムであるが，同様の拡張を PRM に対しても適用することができ

ⓒ 2014 Information Processing Society of Japan

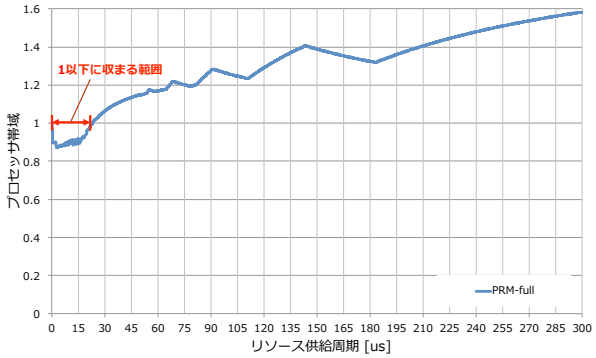


図 8 PRM におけるパラメータ計算結果

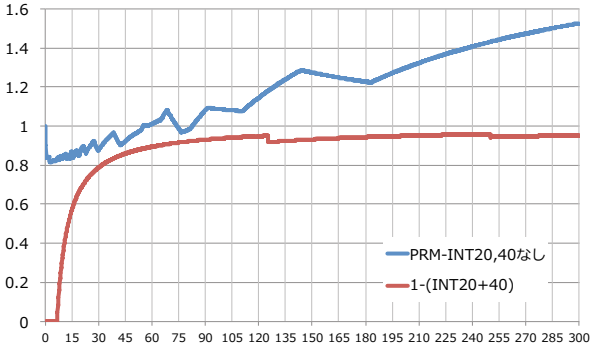


図 9 PRM におけるパラメータ計算結果（INT20 と INT40 を特権割込み処理として指定）

る．しかしながら，先に明らかになったように，PRM では ARINC 653 よりも多くのプロセッサ帯域を必要とするため，今回の適用事例では，適用可能なパラメータの組合せが存在しないという結果になった．例えば，INT20 と INT40 を特権割込み処理に指定し，残りを PRM でスケジュールする場合のパラメータ計算結果を図 8 に示す．この図から，プロセッサ帯域の合計が 1 以下となるリソース供給周期が存在しないことが明らかである．

4.4 考察

スケジューリングの処理オーバヘッドを出来る限り小さくするためには，プロセッサ帯域の合計が 1 以下となる中で，最大のリソース供給周期を選択することが望ましい．適用事例での結果では，まず，ARINC 653 と PRM を適用した場合，それぞれ図 4 と図 8 に示したように，合計のプロセッサ帯域が 1 以下となる中で選択できるリソース供給周期が非常に短かったことから，現実の環境で適用するのが困難であると予想する．それに対して，提案アルゴリズムでは，特に INT20 と 40 の 2 つを特権割込み処理に指定した場合，図 6 に示したように，リソース供給周期を最大 135μs にまで延ばすことができた．この周期は，ARINC 653 の約 3 倍，PRM の約 5.9 倍である．以上より，提案アルゴリズムは，ARINC 653 と PRM に比べて，より現実の環境で適用しやすいと考える．

5. おわりに

本論文では，複数のリアルタイムアプリケーションを単一のプロセッサ上に統合するためのスケジューリングアルゴリズムを提案した．提案したアルゴリズムは，航空機分野の標準規格である ARINC 653 をベースとし，特権割込みを導入することで，高い応答性を実現できるよう拡張したアルゴリズムである．本論文では，アプリケーションが要求するプロセッサ時間の計算に GMF を用いた．GMF によりモデル化できる制御アプリケーションであれば，本論文で用いた方法で設計パラメータを計算できる．

今後は，他の制御アプリケーションを用いた評価，アプリケーションやタスクを切替えるオーバヘッドを考慮した評価，現実の RTOS をベースに提案アルゴリズムを実装することが考えられる．

参考文献

[1] RTCA/DO-297：Integrated Modular Avionics (IMA) Development Guidance and Certification Considerations, RTCA, 2005.
[2] ISO, ISO 26262：Road vehicles - Functional safety - , Part 1〜9, 2011.
[3] S. Baruah, D. Chen, S. Gorinsky, and A. Mok, Generalized multiframe tasks, Real-Time Syst., Vol. 17(1), pp.5-22, 1999.
[4] Airlines Electoronic Engneering Commitee: AVIONICS APPLICATION SOFTWARE STANDARD INTERFACE, (2008).
[5] Lipari, G. and Baruah, S. ”Efficient Scheduling of Real-Time Multi-Task Applications in Dynamic Systems. IEEE Real-Time Technology and Applications Symposium. pp.166-175 (2000).
[6] 松原豊，本田晋也，高田広章，”時間保護のためのタスク起動遅延付き階層型スケジューリングアルゴリズム．”，情報処理学会論文誌，vol.52，No.8，pp.2387-2401 (2011).
[7] 佐藤祐一，松原豊，高田広章，“オーバヘッドを考慮したシミュレータによる階層型スケジューリングアルゴリズムの評価”，第 31 回 EMB 合同研究発表会 (2013).
[8] Shin, I. and Lee, I. ”Periodic Resource Model for Compositional Real-Time Guarantees”, 24th IEEE Real-Time Systems Symposium(RTSS) 2003, pp.2-13 (2003).
[9] Easwaran, A., Lee, I., Sokolsky, O., Vestal, S. “A Compositional Framework for Avionics (ARINC-653) Systems”, Technical Reports, Univ. of Pennsylvania (2009).
[10] Linh T.X. Phan, Meng Xu, Jaewoo Lee, Insup Lee, Oleg Sokolsky. ”Overhead-Aware Compositional Analysis of Real-Time Systems”. 19th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS). pp.237-246 (2013).
[11] Mok, A.K. Xiang Feng, Deji Chen. ”Resource partition for real-time systems”. Real-Time Technology and Applications Symposium, 2001, pp.75-84 (2001).
[12] Timo Kerstan, Daniel Baldin, Stefan Groesbrink. “Full virtualization of real-time systems by temporal partitioning”. OSPERT2010, pp.24-32 (2010).

ⓒ 2014 Information Processing Society of Japan