5A-7

資源制約モデルを用いた CPU 資源制御技術の開発とそのモデルベース開発ツールへの組込み

鈴木 康文† 小川 秀人‡

株式会社日立製作所 中央研究所†,‡

1. はじめに

近年、組込みソフトウェア開発の大規模化や開発期間の短縮が進む中、ソフトウェアの品質保持や開発コスト削減のためにはソフトウェアの再利用が必須となっている。組込みソフトウェアを再利用する上での障害の一つとして、システムのリアルタイム性に関する設計が困難であるという点が挙げられる。ソフトウェアの再利用は、システムを機能ごとに分割した上で各機能をソフトウェアコンポーネントとして開発し、それらを組み合わせることでシステム全体を構築してきた。しかしながら、組込みシステムに顕著な傾向として、CPU の処理能力やメモリ容量等のハードウェア資源が限られているために、単にコンポーネントを組み合わせただけでは性能要求が満たされないケースが多い。製品が持つハードウェア資源の制約や同時に動作する他プログラムとの組合せに応じて、ソフトウェアの実行時間に関する要求(リアルタイム制約)を満たすためのチューニング作業が必要となる。このチューニングのために、プログラムに対して製品毎の改変を加えることがソフトウェアの再利用性の低下を引き起こしている。

このような問題を解決するために、我々はプログラムに対してリアルタイム制約を与えると、その制約を自動的に満たすようにプログラムが実行されるフレームワーク技術を開発している。これによりチューニングのための工数が削減されるのみならず、開発プログラムに対してチューニングコードが入り込まないため再利用性が向上することが期待される。

プログラムが要求された時間内に実行されるためには、プログラムの実行に必要な資源量を見積り、それに応じて適切な資源を配分することが必要である。本研究ではリアルタイム制約を満たすフレームワーク構築のための第一段階として CPU 資源の配分に着目し、CPU 資源制約を与えると、その制約を自動的に満たすようなフレームワークを開発した。また、そのフレームワークをモデルベース開発(MBD)ツールである MATLAB/Simulink®の自動コード生成器に組込み、その性能を評価した。

2. 資源制約モデルに基づく CPU 資源制御技術

2.1 資源制御フレームワークの実現手法

与えられた CPU 資源制約を満たすようにプログラムが動作するためのフレームワークとしては、OS の機能として資源制御機構を提供する手法[1]が実装されている。本研究では、幅広いプラットフォームへ対応するために、

CPU Resource Management Based on Resource-Constraint Models and Its Application to Model Based Development Tools
† Yasufumi Suzuki, Central Research Laboratory, Hitachi, Ltd.
‡ Hideto Ogawa, Central Research Laboratory, Hitachi, Ltd.

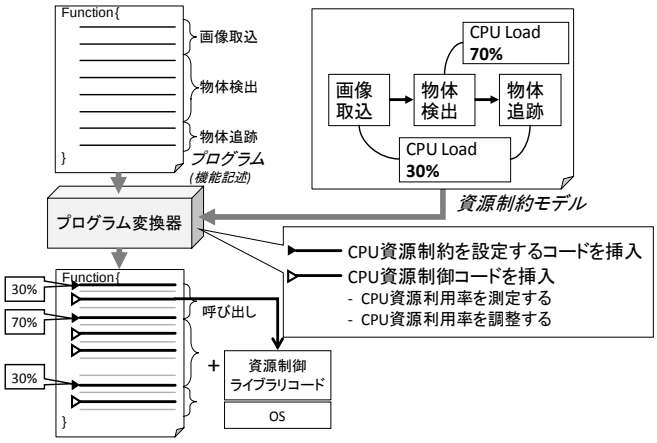

図 1 プログラム変換による CPU 資源制御の実現方法

また、プログラムの実行状況に応じた動的な資源配分の調整を実現するために、ソースコードに対するプログラム変換によって資源制御コードを挿入する手法を用いた。図 1 に、本研究における CPU 資源制御技術の実現手法について示す。

開発者は、ソースコードの各区間に対して何%の CPU 利用率で動作するかを指定した資源制約モデルを記述する。プログラム変換器に両者を入力することにより、自動的に制約を満たしながら動作するような資源制約充足プログラムが生成される。資源制約充足プログラムの生成は、元のプログラムに対して、

- 資源制約が指定された区間の先頭に、その区間の CPU 利用率制約を設定するコードを挿入する
- プログラム中に適当な間隔で後述する CPU 資源制御コードを挿入する

という 2 種類のコードの挿入により実現される。

2.2 CPU 資源の制御手法

プログラム変換器によって挿入される CPU 資源制御コードでは、自タスクの CPU 利用率の測定および自タスク

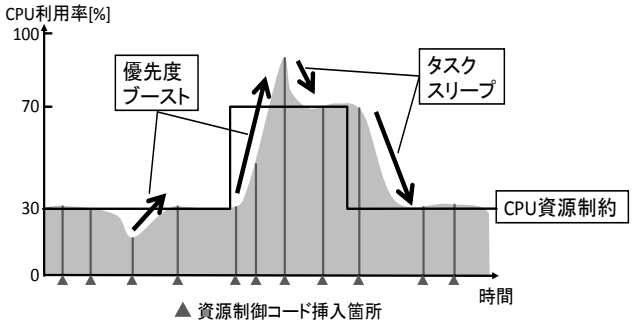

図 2 CPU 資源制約を満たすためのフィードバック制御

の CPU 利用率の調整をおこなう。CPU 資源制御により CPU 利用率が調整される様子を図 2に示す。

タスクは資源制御コードが挿入された各地点において自らの CPU 利用率を計測し、フィードバック制御をすることにより指定された利用率を守って動作する。すなわち、計測した自タスクの CPU 利用率が目標とする利用率に比べて大きい時は自タスクをスリープさせることにより利用率を減少させる。逆に、 利用率が目標値より小さい時は自タスクの優先度を上昇させる(優先度ブースト)ことで利用率の増加を試みる。

両者の制御を実行する CPU 利用率の目標値として異なる値を用いることが可能である。すなわち、スリープ動作を実行する閾値となる利用率(最大 CPU 利用率) と、優先度ブーストを実行する閾値となる利用率(最小 CPU 利用率) とを異なる値を指定することができる。この場合に、CPU 利用率が両者の値の間である場合には、資源制御がおこなわれず、OS のスケジューリングに資源配分を任せることになる。

3. モデルベース開発ツールとの連携

本研究で開発したフレームワークとモデルベース開発ツールとの連携の可能性を実証するために、MATLAB/Simulink® の自動コード生成器である Real-Time Workshop® に対して、構築したフレームワークの組込みをおこなった。

3.1 Simulink 上における資源制約モデル記述

MATLAB/Simulink® では、機能ブロックをつなぎ合わせたデータフローモデルを作ることでプログラムモデルを記述する。また、サブシステムを用いることでモデルを階層的に記述することが可能である。そこで、図 3 に示すように、サブシステム内に資源制約指定ブロックを置くことにより、そのサブシステム内で記述されるプログラムを実行する際の CPU 利用率制約を指定することとした。

3.2 プログラム変換の実現方法

CPU 資源制御コードは、プログラム中のあらゆる場所に挿入される。一般に、挿入箇所が多いほどオーバヘッドが大きくなるが、少ないほど資源利用率のチェックがされない期間が長くなり、精度が落ちるというトレードオフがある。また、挿入箇所の間隔は実行時における時間間隔が均等になるように配置されることが望ましい。

Real-Time Workshop では、あらかじめブロック単位でソースコードが用意されており、モデルで記述されたブロック間の結合に従ってブロックごとのソースコードを組み合わせることにより全体のソースコードを生成している。可能な限り細かく CPU 資源制御コードを挿入する

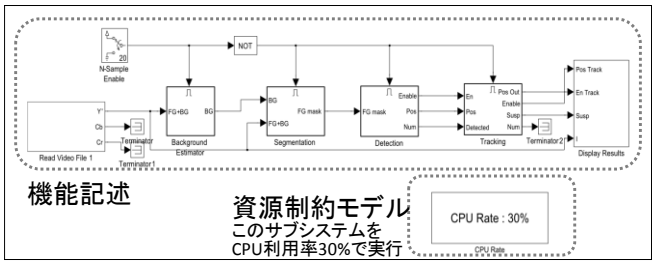

図 3 Simulink 上での資源制約モデル記述

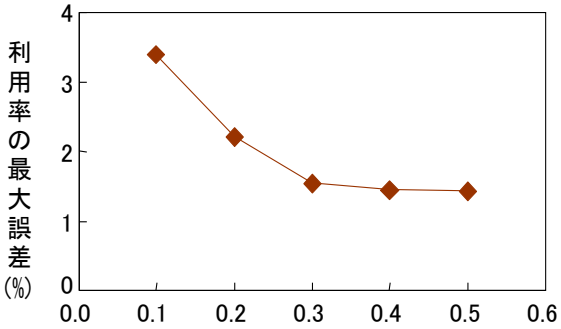

図 4 CPU 資源制御のオーバヘッドと精度の関係

には、ブロックとブロックの境界毎にコードを挿入すれば良い。本研究では Real-TimeWorkshop がブロックに対応するコードを生成する部分を改造し、ブロックの境界毎に CPU 資源制御コードを挿入するようにした。

しかし、各ブロックのプログラムの計算量はブロック毎に様々であり、計算量の小さいブロックの前後では短い時間間隔で資源制御コードの実行が繰り返され、オーバヘッドが大きいという問題がある。オーバヘッドの増大を防ぎ、資源制御の実施間隔の平準化するために、資源制御コードが実行されてから一定時間内においては、再び資源制御コードが呼び出されても資源制御処理をおこなわないこととした。

4. 結果と評価

本研究で開発した機能を組み込んだ Real-Time Workshop を用いて生成されたコードを用いて評価を実施した。アプリケーションモデルとして Simulink の Video and Image Processing Blockset に付属する動画像からの物体検出デモのモデルを使い、生成されたコードを Linux OS 上で動作させた。開発したフレームワークを用いることによるオーバヘッド(計算量の増加)と、資源制御の精度とを評価した結果を図 4に示す。

前述のように CPU 資源制御の精度と計算量オーバヘッドの間にはトレードオフ関係があることが確認された。0.4%程度のオーバヘッドで 1.5%以内の精度の CPU 資源制御が実現されており、実用上、充分な精度と低オーバヘッドであると言える。

5. 結論

与えられた CPU 資源制約を自動的に守るようにプログラムを実行するフレームワークを構築し、MBD ツールへの組込みをおこなった。その結果、1.5%以内の精度の CPU 資源制御を低オーバヘッドで実現した。今後は、本技術と、プログラムの実行に必要な資源量を見積る技術とを組合せることにより、リアルタイム制約を自動的に満たすフレームワーク構築をめざす。

参考文献

[1] M. Sugaya et al., “Accounting system: A fine-grained CPU resource protection mechanism for embedded system,” in *Proc. Intl. Symp. Object and Component-Oriented Distributed Computing (ISORC)*, Apr. 2006.