社団法人 電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report

リソース制約を持つ複数タスク動作仕様におけるタイムバジェット最適化の一手法

倉田 和哉† 百々 太市† 中田 明夫‡

†‡広島市立大学 大学院情報科学研究科 システム工学専攻

〒731-3194 広島市安佐南区大塚東 3-4-1

E-mail: †{kurata, dodo}@sos.info.hiroshima-cu.ac.jp, ‡nakata@hiroshima-cu.ac.jp

あらまし 性能とリソースの制約が共に厳しく，高い信頼性を要求される組込みソフトウェア開発においては，設計モデルの段階で時間制約やリソース制約を考慮した最悪時の性能検証を行うことが有用である．我々は従来，複数タスク動作仕様と各タスクが要するリソースの割り当て及びリソーススケジューリング方針が与えられたとき，性能要求を満たすか否かの検証を行う手法の提案を行ってきた．特に設計初期段階では，性能を満足するために各タスクがどれだけ実行時間を使って良いかを適切に決定できれば，性能要求を満たしコストや消費電力を抑えたシステム設計に有用である.そのような各タスクへの実行時間の配分をタイムバジェットと呼ぶ．本研究では，性能要求を満たすモデルに対し，同性能で各タスクのタイムバジェットを最適化する手法を提案する．

キーワード リアルタイムシステム，性能検証，組込みソフトウェア，設計モデル

A Time Budget Optimization Method for Multi-Task Behavioral Specifications with Resource Constraints

Kazuya KURATA† Taichi DODO† and Akio NAKATA‡

†‡Department of Systems Engineering, Graduate School of Information Sciences, Hiroshima City University

3-4-1 Ozuka-higashi, Asaminami-ku, Hiroshima-shi, Hiroshima 731-3194, Japan

E-mail: †{kurata, dodo}@sos.info.hiroshima-cu.ac.jp, ‡nakata@hiroshima-cu.ac.jp

Abstract In the development of embedded software, where performance and resource requirements must be highly optimized and high-level reliability is required, it is useful to verify the worst case performance considering time and resource constraints in the early phase of the design process. We formerly proposed a performance verification method of multi-task specification with preemptive resource scheduling given multi-task behavioral, resource allocation to each task, and resource scheduling policies. Especially in the early phase of the design process, it is useful to allocate the execution times among all tasks for the system to satisfy its performance requirement properly. Such an allocated execution time for a task is called a ‘time budget.’ In this paper, we propose a method to optimize the time budgets for all tasks while keeping the model satisfying its performance requirements.

Keyword real-time system, performance verification, embedded software, design model

1. まえがき

機器に組み込まれるソフトウェアである組み込みソフトウェアにおいては，コストや消費電力などの厳しいリソース制約のもとでの実時間性の実現を要求されることが多く[1]，機能的要求に加えて性能などの非機能的要求の達成が重要となる．しかし，伝統的なソフトウェア開発手法においてはソフトウェア機能の正しさに焦点が置かれており，設計の途中で性能が大きく変化することを考慮していない．そのような開発手法においては性能問題が開発プロセスの後半で見つかることが多く，設計の見直しなど手戻りが生じる．そこで．開発プロセスの早期の段階で性能検証を行うことが有用である．性能検証とはソフトウェアシステムがユーザの性能目標を満足するか否かを性能モデルに基づき予測し，検証することを指す．性能検証を行う手法としてはソフトウェア仕様モデルからソフトウェア性能検証モデルへ変換し，性能検証を行う方法が提案されている[2]．ソフトウェア性能モデルには，待ち行列ネットワークなどの確率的モデル[3]と時間ペトリネット[4]などの確定的モデルに分類される．確率的モデルはシステムの平均性能を考慮するのに対して，時間ペトリネットは最悪時(および最良時)の性能を考

慮する．性能とリソース制約がともに厳しく，高い信頼性を要求される組み込みソフトウェアを対象とする場合，最悪時に性能要求を満たせるか否かの検証が重要になってくる．

我々は従来，複数タスク動作仕様とリソース割り当て図(タスクグラフ)を優先権付きストップウォッチペトリネット(PrSwPN)に変換し，そのモデルを用いてスループット(単位時間当たりの処理能力) 要求を満たすか否かの検証を行う手法を提案している[5]．しかし，従来手法ではスループット要求を満たすか否かの判断のみ行うため，検証結果をもとにモデルを改良することが困難であった．特に設計初期段階における各タスクの最悪実行時間は，実装前の段階では大まかな見積もりに基づく各タスクへの実行時間の配分と呼ぶべき性質のものである.これを本研究ではタイムバジェット(time budget)と呼ぶ.システム全体のスループットやレイテンシ（個々のデータが入力されてから出力されるまでの遅延）に関する制約を満たすように各タスクのタイムバジェットを適切に決定し，各タスクの実装段階においてはタイムバジェットの範囲内で実行できるようにプログラムコードおよびプロセッサやネットワークなどのハードウェアプラットフォームの性能を決定することにより，性能要求を満たすシステムの効率的な設計が可能となる.本研究では,[5]の性能検証手法に対して，モデルのレイテンシ性能を検証するよう従来手法を改良し，与えられたレイテンシ要求に対する余裕時間を各タスクに効果的に配分することにより，同性能で各タスクのタイムバジェットを最適化する手法を提案する．

2. 諸定義

本章では，本研究が対象とするモデルや解析手法に関する定義を述べる．

2.1. リアルタイムシステム

リアルタイムシステムとは，設定された時間通りに動作しなければならないコンピュータシステムのことをいう．通常，リアルタイムシステムではイベント(event)の発生ごとにタスクがリリースされ，定められたサービスを実行しなければならない．そしてある時刻までにサービスを完了しなければならない．イベントがリアルタイム処理系に発生すると，このイベントに関連づけられたタスクが実行を行う．また，タスクの実行のためには実行に必要となるリソースが全て使用可能であることが必須である.図 3 にタスクの実行状況を示す．

図 2.1.1 タスクの実行状況

イベントが発生すると，そのイベントに関連づけられたタスクが到着(arrive)という．このことをタスクがリリース(release)されると呼び，この時点を矢印“↑”で示す．四角“■”の部分はタスクの処理を表している．start はタスクの処理の開始時刻，final はタスクの処理の完了時刻を表す．

タスクの最悪実行時間とはタスクが実行を開始してから実行を完了するまでにかかる一番長い時間のことをいう，図 3 においては start から final までの時間が最悪実行時間に対応する．タスクの最悪応答時間とはタスクがリリースしてから実行を完了するまでにかかる一番長い時間のことを言う．図 3 においては release から final までの時間が最悪応答時間に対応する．release から deadline までの時間のことを相対デッドラインという．一方，現在時刻にタスクの相対デッドラインを加えたものをタスクの絶対デッドラインという．タスクの処理の完了時刻 final がデッドライン以下であるとき，タスクはデッドラインを満たすという．

2.2. スケジューリング方式

スケジューリング方式とは，一般に尐数のリソースを共有する複数のタスク群において，すべてのタスクがデッドラインを満たすように，各タスクをどのような順番でいつからいつまで実行するかを決定する方式である．ここでリソースとしては，計算リソース（プロセッサ）のみならず，通信リソース（ネットワークや共有変数など）も含めて考慮する．本論文では以下の 2 つのスケジューリング方式を扱う．

① 早い者勝ち

早い者勝ち(FCFS: First Come First Served)とは先に到着したタスクが即座にリソースを獲得して実行するというスケジューリング方式である．

② 固定優先度スケジューリング

固定優先度スケジューリング(FP: Fixed Priority scheduling)とは，各タスクに

固定の優先度を割り当て，複数のタスクがリソースについて競合した場合には割り当てられた固定優先度の大きいタスクがリソースを獲得し実行するというスケジューリング方式である．

2.3. 優先権付きストップウォッチペトリネット

$\mathbb{N}$を 0 を含む自然数全体の集合，$\mathbb{R}^+$を非負実数全体の集合とする．ペトリネット(Petri Nets)は並列的・非

同期的・分散的なシステムを表すための数学的モデルであり，5 つの組$PN = (P,T,F,W,M_0)$で定義される．ここで，P はプレースの有限集合，T はトランジションの有限集合，$F \subseteq (P \times T) \cup (T \times P)$ はアークの有限集合，$W: F \mapsto \mathbb{N}$ はアークへの重みづけ関数である．アーク $(p,t) \in F$ に対して，p を t の出力プレースと呼ぶ．各プレースには一般に複数のトークンを配置することによりシステムの状態を表現する．各プレースにおかれたトークンの数を返す関数$M_0: P \mapsto \mathbb{N}$ を初期マーキングと呼ぶ．マーキング M において，トランジション $t \in T$ のすべての入力プレース p について，$M(p) \geqq W(p,t)$ のとき，tは発火可能であるという．またトランジション t が発火するときは，t が発火可能なマーキンにおいて，t の任意の入力プレース p から重み $W(p,t)$の数だけトークンを取り除き，t の任意の出力プレース p' に重み $W(t,p')$ の数だけトークンを追加する動作のことであると定義する．トランジションの発火によりシステムの状態遷移が定義される．ペトリネット PN の各トランジション$t \in T$ に，発火可能になってから発火するまでの時間の下限(最少発火遅延)$d_{min}: T \mapsto \mathbb{R}^+$ および，上限(最大発火遅延) $d_{max}: T \mapsto \mathbb{R}^+ \cup \{\infty\}\left(d_{max} = \infty \text{ならば上限なし}\right)$ を付与したペトリネットを時間ペトリネット（Time Petri Nets）と定義する．時間ペトリネットにおいては，各トランジション t は t が最後に発火可能になってからの経過時間 $\theta(t) \in \mathbb{R}^+$を保持するものとし(発火可能でないなら $\theta(t) \in -\infty$ とする)，t が発火不能にならない限り，$d_{min} \leqq \theta(t) \leqq d_{max}$ を満たすいずれかの時刻に t は発火しなければならないと定義される．時間ペトリネットに，トランジション間の優先権$\succ \subseteq \mathrm{T} \times \mathrm{T}$ を付与したものを優先権付き時間ペトリネット(PrTPN)[ ]と定義する．優先権付き時間ペトリネットでは，2 つのトランジション$t_1, t_2 \in \mathrm{T}$ が同時に発火可能なとき，もし $t_1 \succ t_2$ ならば，t_1 のみが発火できるものとする．

優先権付き時間ペトリネットに，ストップウォッチアーク $S_w \subseteq T \times P$ およびその重み付け関数 $W_{sw}: S_W \mapsto \mathbb{N}$を付与したものを優先権付きストップウォッチペトリネット(PrSwPN)と定義する．優先権付きストップウォッチペトリネットでは，マーキング M においてトランジション t のすべてのストップウォッチアーク(t,p) に対して$M(p) \geqq W_{sw}(t,p)$ ならば，t は最後に発火可能になってからの時間 $\theta(t)$を更新し続ける．さもなければ，t は $\theta(t)$の値の更新を停止する．

3. 性能検証手法

3.1. タスクグラフ

[5]の検証手法では，複数タスク動作仕様とリソース割り当て図をモデル化するために，タスクグラフを用いている．タスクグラフとは，順序関係を持つタスク同士を矢印でつなぎ，有向非閉路グラフとして表現したものである．タスクグラフと等価な振る舞いを行い，かつ，スループット要求を満たさなければ特定の動作を行う PrSwPN を作成し，スループット要求を満たすか否かの検証を行う．

タスクグラフ TG は 9 つの組 TG =(T, R, C, a, c, d, rr, sc) である．ここで，T はタスクの有限集合，R はリソースの有限集合，C は制御ノードの有限集合，$\rightarrow \subseteq (T \times C) \cup (C \times T)$ は依存関係，$a: C \mapsto \{input_\lambda, par, join, choice, endchoice, output\}$(ただし,$\lambda \in R^+$) は制御ノードの種類を返す関数，$c: T \mapsto R^+$ は各タスクの最悪実行時間を返す関数，$d: T \mapsto R^+$ は各タスクの相対デッドラインを返す関数，rr: $T \mapsto 2^R$ は各タスクの実行に必要なリソースの集合を返す関数，sc : $R \mapsto \{FIFO, FP_\gamma\}$ は各リソースのスケジューリング方式，$\gamma: T \mapsto \mathbb{N}$ は各タスクへの優先度割り当て関数である．$N = C \cup T$ とし，$n \in N$ をタスクグラフ TG のノードと呼ぶ．G = (N,→) は DAG(閉路の無い有向グラフ) となっているものとする．$n_1, n_2 \in N$ のとき，$(n_1, n_2) \in$を $n1\ !\ n2$ と書き，n_1はn_2 に先行する，n_2はn_1 に後続すると呼ぶことにする．

表 3.3.1 に制御ノードの動作意味を，図 3.3.1 にタスクグラフの例を示す．

表 3.3.1 制御ノードの動作意味

	先行タスクの待ち条件	後続タスクの起動方法
input λ	なし	一定周期 λ
choice	一つが終了するまで待つ	いずれか一つを選択
par	一つが終了するまで待つ	全てを起動
endchoice	いずれかが終了するまで待つ	一つ
join	すべてが終了するまで待つ	一つ
output	一つが終了するまで待つ	なし

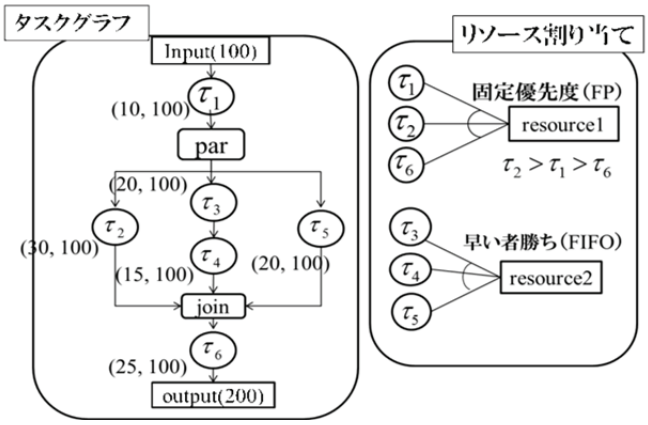

図 3.3.1 タスクグラフの例

3.2. タスクグラフから PrSwPN への変換

文献[5]の手法に従い，与えられたタスクグラフ *TG* から，*TG* の各動作（任意のタスク τi のリリース，実行開始，実行中断，実行再開，実行終了，リソースの獲得・解放）の実行系列の集合（時間付きトレース集合）と，対応するトランジションの実行時刻も含めた発火系列の集合が等しいような PrSwPN に変換する．この変換は，まず，タスク，スケジューラなどのシステムの構成要素それぞれに対して，対応する PrSwPN による部分モデルを構成し，それらを結合することにより行っている．部分モデルの例として，図 3.2.1 に PrSwPN で表したタスクを示す．タスクの起動が要求されると，まず起動要求プレースに（外部から）トークンが置かれ，それにより起動トランジションが直ちに発火し，リソース要求プレースおよび起動中プレースにトークンが移動する．リソース要求プレースにトークンが置かれると，スケジューラモジュールは他のタスクからの当該リソース要求を適切に調停し[1]，リソースを獲得できたらならば実行可能プレースおよびリソース獲得中プレースにトークンを移動させる．リソースを獲得しタスクを実行開始してから最悪実行時間（＝タイムバジェット）経過後に，実行完了トランジションが発火し，トークンがリソース返還プレースおよび次状態プレースに移動して実行終了となる．実行中に他のタスクから横取り（プリエンプション）があった場合，スケジューラモジュールによってリソース獲得中プレースからトークンが引き去られ，ストップウォッチアークにより実行完了トランジションの時間計測が停止する．なお，起動中プレースにトークンが置かれてからの経過時間をデッドライントランジションが計測しており，デッドラインを超過するとデッドライン超過トランジションが発火し，エラー状態（タスク失敗）となる．

[1] スケジューラモジュールの PrSwPN によるモデル化の詳細は紙面の制約により割愛する

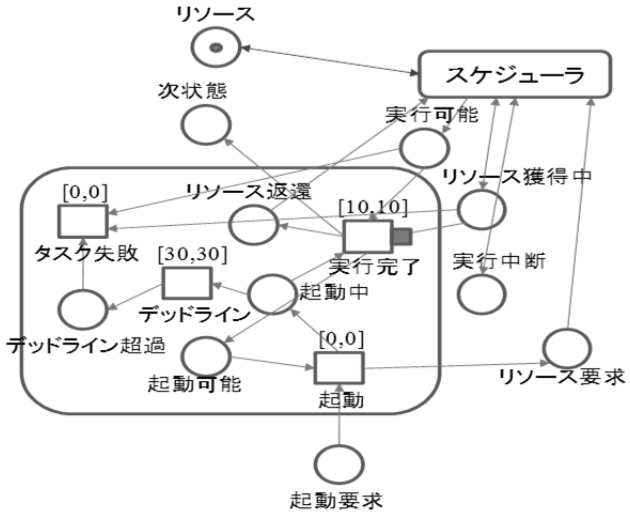

図 3.2.1 PrSwPN で表したタスク

3.3. スループット検証

性能検証の手法として，性能検証モデルに対しスループット検証を行う．スループット要求が満たされていない状態を検出するアサーションを性能検証モデルに追加してスループットを検証する．入力が入ってくるプレースに対して 2 個以上トークンがたまるとエラー状態になるようなアサーションをつける．具体例を図 3.3.1 に示す．error プレースはエラーの状態を表すプレースで，トークンがある場合，現在エラーであることを示す．assertion トランジションはエラー処理を行うトランジションを表している．この例では，p0 プレースに input トランジションから 100ms ごとにトークンが入ってくる．つまり，スループットは 10 単位データ/秒である．もし p0 プレースにトークンが 2 つ以上たまれば，100ms 以内に処理できていないことになる．その場合 assertion トランジションが即座に発火し，error プレースにトークンが移動する．

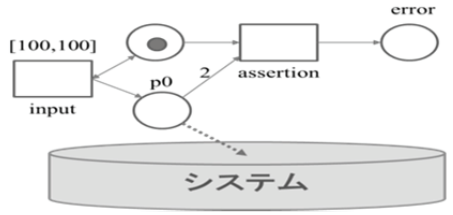

図 3.3.1 スループット検証を行うための PrSwPN 表現

4. 提案手法

入力から出力までの最悪応答時間を求めることで，得られたデッドラインまでの余裕時間を各タスクのタイムバジェットに配分する．配分は，同じリソースを共有するタスクの最悪実行時間を同じ倍率増加させる

ことで行う．最適化は，すべてのタスクの最悪実行時間の合計が最大になるような倍率の組み合わせを求めることで行う．

4.1. タスクの最悪応答時間の計測

[5]でのタスクから PrSwPN への変換手法に，タスクの最悪応答時間を計測するためのプレースとトランジションを追加する．モデルの例を図 4.1.1 に示す．例のタスクの最悪実行時間は 5 秒，デッドラインは 10 秒とする．簡単のため一部のプレース，トランジションは省略している．タスク内の起動トランジションが発火すると，計測中プレースにトークンが移動する．計測可能プレースにトークンがある場合，5 秒経過トランジション，6 秒経過トランジションの順に順次発火する．10 秒経過トランジションが発火する前に実行完了トランジションが発火すると，計測可能プレースからトークンが奪われ，計測が中断される．10 秒経過トランジションが発火すると，デッドラインオーバープレースにトークンが移動し，タスク失敗トランジションが発火する．タスク失敗トランジションはタスクが処理に失敗したことを表している．

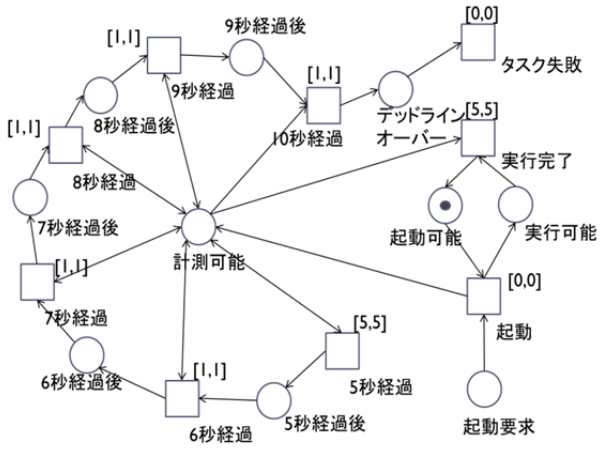

図 4.1.1 PrSwPN で表したタスク

4.2. 入力から出力までの最悪応答時間の計測

入力から出力までの処理(以下，パスと呼ぶ)にかかる最悪の場合の時間を計測するためのプレースとトランジションの組を追加する．システムに入力が入ると計測を開始し，システムから出力があると計測を停止する．モデルの例を図 4.2.1 に示す．入力複数の入力に対応するため，計測部分を複数用意する．同時に実行されるパスの最大数は，パスのデッドラインを pd，入力周期をλとすると，pd/λであるため，計測部分は$\lfloor pd/\lambda \rfloor$個必要となる．

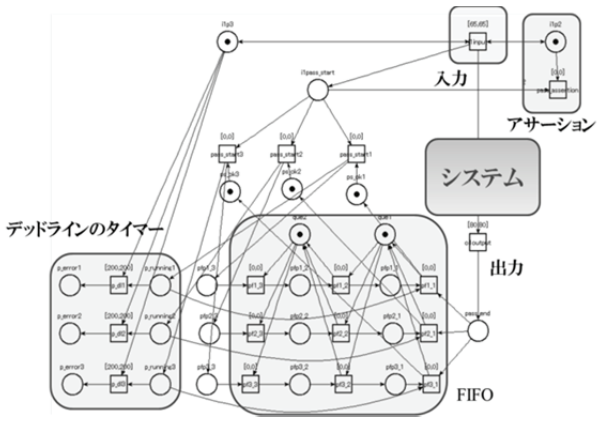

図 4.2.1 パスの最悪応答時間計測モデル

4.3. 複数タスク動作仕様の最適化

最適化は，すべてのタスクの最悪実行時間の合計が最大になるような組み合わせを求めることで行う．同じリソースを共有するタスク群のうち，一部のタスクのみのタイムバジェットを増やすと，増やさなかったタスクがリソースをとることが難しくなり，リソース共有の飢餓状態が発生する恐れがある．そのため，同じリソースを共有するタスクの最悪実行時間を同じ倍率で増加させる.以下に最適化の手順を示す.ここで，$r \in R$ $inc(r)$はリソース$r \in R$を共有するタスクの最悪実行時間の増加率，f はタイムバジェッティングの精度を表すパラメータ， $passWCRT(TG)$はタスクグラフ TG のパスの最悪応答時間とする．$inc(r)$の初期値は 0 である．

1. すべてのリソース$r_i \in R$において，それぞれのリソースを共有するタスク$t_{ij} \in T$の最悪実行時間 $c(t_{ij})$を $c'(t_{ij})$に置き換えたものを新しいタスクグラフ TG_i とする．$c'(t_{ij})$は以下のように求める．
   $c'(t_{ij}) = c(t_{ij}) + c(t_{ij})*(inc(r_i) + f)$
2. すべての TG_iを性能検証モデルに変換し，変換したモデルに対し性能検証を行い，$passWCRT(TG_i)$を求める．
3. 2.において，もしすべての TG_iが性能要求を満たさないならば，5.に移る．
4. すべての $passWCRT(TG_i)$のうち最小の値となる TG_iを選び，$incr_i$に fを足し，1.に戻る．
5. すべてのタスク$t_i \in T$に対する $c(t_i)$を $c'(t_i)$に置き換えたものを新しいタスクグラフ TG' とし，出力する．$c'(t_i)$は以下のように求める．
   $c'(t_i) = c(t_i) + c(t_i)*(inc(rr(t_i)) + f)$

一般に，タスクの最悪実行時間を増やすと，その分リソースを使用する時間が増えるので，ほかのタスクのリソース使用時間を圧迫してしまう．4.において，パ

スの最悪応答時間が最小になるような *TG* を選択しているのは，最悪実行時間増加による他のタスクへの影響が最も少ないと考えられるためである．

5. 実験

図 3.3.1 の例題について，性能検証モデルへの変換を行い,そのモデルに対しスループット検証を行った．また，検証結果をもとにモデルの最適化を行った．なお，入力周期は 10[入力/s]，パスのデッドラインは 200[ms]，増加率 f は 0.5[%]とする．

5.1. 検証環境

検証には PrSwPN 検証ツール TINA[6]を用いる．TINA は PrSwPN をグラフィカルに描写することができ，作成した PrSwPN をテキストファイルに変換することもできる．ランダムシミュレータでは，トランジションの発火時間をランダムに選んで，解析を行う．到達可能性解析では，すべてのマーキングを調べ，PrSwPN がデッドロックかどうかを解析することができる．つまり，TINA はデッドロックの検証を行う機能を持っている．

5.2. 検証結果

最適化前，最適化後それぞれについてのモデルの検証結果と各タスクの最悪実行時間を表 5.2.1，表 5.2.2 に示す．

表 5.2.1　最適化前のモデルの検証結果

	最適化前	最適化後
スループット要求を満たすか否か？	Yes	Yes
検証時間[ms]	140.0	1513.0
状態数	1583	8302
パスの最悪応答時間[ms]	90	197

表 5.2.2 最適化前のモデルの各タスクの最悪実行時間と最悪応答時間

タスク名	最適化前の最悪実行時間	最適化後の最悪実行時間
τ 1	10	24
τ 2	30	73
τ 3	20	27
τ 4	15	20
τ 5	20	27
τ 6	25	61

5.3. 考察

resource1，resource2 を共有するタスクの最悪実行時間をそれぞれ 146.0%，39.5%増加させても，最適化前と同性能のモデルを得ることができた．先行研究[5]の変換手法で同モデルを変換した場合，状態数は 100 程度であったが，本実験では，パスの最悪応答時間の計測を行うプレースとトランジションの組を追加したことによりトークンの移動が複雑になったため，状態数が増大したものと考えられる．しかし，それでも数秒程度で検証が完了している．従って，本実験で扱った程度の例題規模であれば,提案手法により性能検証，モデルの最適化が行えるといえる．

6. 結論

本研究では，プリエンプティブスケジューリングでリソースを共有し，分岐や並列などの制御構造を持つ複数タスク動作仕様の入力から出力までの最悪応答時間を既存の性能検証手法により求め，得られたデッドラインまでの余裕時間を各タスクのタイムバジェットに効果的に配分することにより，性能要求を満たしつつもタスクのタイムバジェットを最適化する手法を提案した．本研究では性能要求の検証に確定的な時間を取り扱う PrSwPN を用いているため，最悪の場合の性能を検証することが可能であり，組込みソフトウェアなどの設計初期段階での性能検証に役立つと思われる．

今後の課題としては，本研究で行ったような発見的な方針の最適化ではなく，遺伝的アルゴリズムや焼きなまし法などを適用した方針で最適化を行うことである.また,複数タスク動作仕様をラウンドロビンや EDF といったスケジューリング方式にも対応させることや,実行に必要なリソースが複数あるタスクに対応することなども今後の課題として挙げられる．

文　献

[1] Henzinger, T.A. and Sifakis, J., ”Embedded Systems Design Challenge” Proc. of 14th Int. Symp. on Formal Methods (FM 2006), Lecture Notes in Computer Science, Vol.4085, Springer Verlag, pp.1-15, 2006.

[2] Balsamo, S., Marco, A.D., Inverardi, P. and Simeori, M., “Model-Based Performance Prediction in Software Development: A Survey”, IEEE Trans. on Softw. Eng., vol30, No.5, pp.295-310, 2004

[3] Balsamo, S. and Marzolla, M., ”Performance evaluation of UML software architectures with multiclass Queueing Network models”, Proc. of 5th Int. Workshop on Software and Performance, Advancing Computing as a Science and Profession, pp.37-42 ,2005.

[4] Berthomieu, B., Peres, F., and Vernadat, F., “Model Checking Bounded Prioritized Time Petri Nets”, Proc. of 5th Int. Symp. on Automated Technology for Verification and Analysis (ATVA 2007), Lecture Notes in Computer Science, vol.4762, pp.523-532, 2007.

[5] 百々太市，中田明夫，”プリエンティブスケジューリングによりリソースを共有する複数タスク動作仕様の性能検証” ,組込みシステムシンポジウム(ESS2010)論文集, pp.107－112, 2010.

[6] Berthomieu, B. and Vernadat, F. “Time PetriNets Analysis with TINA”, Proc. of 3rd Int.Conf. on the Quantitative Evaluation of Systems(QEST 2006), IEEE Computer Society Press, 2006.