Vol.2021-ARC-244 No.21
Vol.2021-SLDM-194 No.21
Vol.2021-EMB-56 No.21
2021/3/25

並列実行タスク間の干渉を考慮したミックスドクリティカリティシステム向けスケジューリング最適化

東山知彦[1]　曾剛[2]　高田広章[2]　外山正勝[1]

概要：近年，車載分野などを中心に，従来分散していた機能を 1 つのコントローラに統合する流れがある．このようなコントローラでは，処理の増加に対応するためマルチコアが採用されることが多い．マルチコアでは，並列に実行(Co-run)するタスク間での共有メモリやメモリバスなどの共有ハードウェアリソース競合による性能劣化(干渉)が生じる場合がある．リアルタイムシステムにおいてはこのような性能劣化を考慮し，可能な限り干渉を回避するスケジューリング設計を行うことが課題となる．また，機能統合時の別の課題として，重要度の異なるタスク混在時のスケジューラビリティ向上が挙げられる．重要度の高いタスクは厳密な最悪実行時間を設定する必要があるため，その影響によりシステム全体のスケジューラビリティが過度低下してしまう．この問題を解決する手段として，Mixed Criticality Scheduling という考え方が提唱されている．Mixed Criticality Scheduling のスケジューラブル判定基準を満たすようにスケジューリング設計することで，リアルタイム性を保証しつつ，スケジューラビリティを向上させることが可能となる．そこで本稿では，Co-run タスクの影響による性能劣化を可能な限り回避すること，Mixed Criticality Scheduling スケジューラブル判定基準を満たすこと，という 2 つの課題を解決可能なスケジューリング設計手法を提案した．提案手法のスケジューラビリティを評価した結果，2 コアの時に平均 4.3%，4 コアで 20.8%のスケジューラビリティ改善効果が得られることを確認した．

Keywords: Mixed Criticality System, Mixed Criticality Scheduling, Co-run, Multicore, Interference, Linear Programming, Task Scheduling, Schedulability

1. はじめに

近年，車載分野などを中心に，コスト削減を目的とし，従来分散していた機能を一つのコントローラに統合する流れがある．このようなコントローラでは，機能統合化による処理量の増加に対応するため，マルチコアプロセッサの採用が進んでいる．マルチコア環境では，並列に実行(Co-run)するタスク間での共有メモリやメモリバスなどの共有ハードウェアリソース競合による性能劣化が生じる場合がある[1][2]．文献[1]では，Raspberry Pi3 Model B+[3](ARM Cortex-A53)上で負荷プログラムを実行した場合，最悪ケースで Co-run タスクにより性能が約 1/160 に劣化したことが報告されている．また，文献[2]では，PC(Core-i7 6500U)上でベンチマークプログラムを実行した場合，最悪ケースで Co-run タスクによる性能が約 2/3 に劣化したことが報告されている．リアルタイムシステムにおいてはこのような性能劣化を考慮し，可能な限り干渉を回避するスケジューリング設計を行うことが課題となる．

機能統合化によるリアルタイム設計のもう 1 つの課題として，重要度の異なるタスク混在時のスケジューラビリティ向上が挙げられる．重要度(クリティカリティレベル)の高いタスクは，厳密な最悪実行時間を想定し CPU 時間を割り当てる必要がある．一方，重要度の低いタスクはデッドラインミスもある程度許容されるため，重要度の高いタスクに最悪実行時間 CPU を割り当てることは過剰な要求を課していることになる．例えば，デッドラインミスの発生確率が 10%以内であればよいタスク(低重要度タスク)と，デッドラインミスが許さないタスク(高重要度タスク)が混在するシステムを考える．問題を簡単にするため，システムはシングルコアであるとする．また，高重要度タスクが 0.1%の確率で CPU 使用率 100%消費するとする(一般的に最悪実行時間は平均的な実行時間に比べて極端に長く，かつ発生頻度は低い場合が多くこのような仮定は現実的である)．この時，低優先度タスクは 0.1%の確率でデッドラインミスとなり，デッドラインミスの確率 10%以下という要求を満たしている．それにもかかわらず，高優先度タスクの実行時間を最悪実行時間と考えると 2 タスクの合計 CPU 使用率が 100%を超えるため，一般的なスケジューリング理論ではスケジュール不可と判定される．

この問題を解決する手段として，Mixed Criticality Scheduling という考え方が提唱されている[4][5]．Mixed Criticality Scheduling のスケジューラブル判定基準を満たすようにスケジューリング設計することで，リアルタイム性を保証しつつ，スケジューラビリティを向上させることが可能となる．よって，本稿では，以下の 2 点を満たすスケジューリング設計手法を提案する．

- Co-run タスクの影響による性能劣化を可能な限り回避しスケジューラビリティを向上させること
- Mixed Criticality Scheduling のスケジューラブル判定基準を満たすこと

2. Mixed Criticality Scheduling

異なる重要度レベル(クリティカリティレベル，以降 CL とも表記)を持つタスクが混在するシステムを Mixed Criticality System と呼ぶ[4][5]．Mixed Criticality System では一般的にタスクiは次のようにモデル化される．

$$\tau_i = (T_i, D_i, \boldsymbol{C_i}, L_i)$$

1 三菱電機株式会社
Mitsubishi Electric Corporation.
2 名古屋大学
Nagoya University

Vol.2021-ARC-244 No.21
Vol.2021-SLDM-194 No.21
Vol.2021-EMB-56 No.21
2021/3/25

T_iは周期，D_iはデッドライン，L_iはクリティカリティレベルを表す．$\boldsymbol{C_i}$はクリティカリティレベル付けされた複数の実行時間の値の集合である．クリティカリティレベルXに対応する実行時間を$C(X)$と表す．ただし，実行時間には以下のような規則に従う．

$$X2 > X1 \Rightarrow C(X2) > C(X1)$$

タスクセットの例を表 1 に示す．システム全体で 3 つの CL が定義されており，H(High)が最も高く，L(Low)が最も低く，M(Middle)がその中間であると定義している．

Mixed Criticality Scheduling は Mixed Criticality System 向けのスケジューリング手法である．Mixed Criticality Scheduling では以下を満たしたときに，スケジューラブルと判定する[7].

スケジューラブル判定の定義：

全てのタスクがクリティカリティレベル X に対応する実行時間以内で完了した場合，タスクの CL が X 以上のタスクはデッドライン以内に完了する．

高い CL のタスクは全てのタスクが高い CL に対応する実行時間を要した場合，つまりより悲観的な実行時間で実行された場合でも，デッドラインを満たすことが保証される．一方，低い CL のタスクは全てのタスクが低い CL に対応する時間を要した場合はデッドラインを満たすことが保証されるが，それ以上の実行時間を要した場合はデッドラインミスを起こす可能性がある．

表 1：Mixed Criticality System のタスクセット例

タスク名	T	D	L	$C(H)$	$C(M)$	$C(L)$
A	5	5	H	5	4	2
B	5	5	H	3	2	1
C	5	5	M	—	3	1
D	5	5	L	—	—	3

3. ベーススケジューリング方式

3.1 TDMA スケジューリング

1 章で述べた通り，本稿では Co-run タスク間の干渉を制御可能な方式を目指す．Co-run の発生を制御するためには各タスクの実行時刻を制御する必要がある．そこで TDMA(Time Division Multiple Access)スケジューリング[6]を採用することとした．TDMA スケジューリングは，タスクに CPU を割り当てる時間(タイムスロット)を事前に規定し，そのタイムスロットではそのタスクが排他的に CPU を使用するスケジューリング方式である．TDMA スケジューリングの場合，タイムスロットが固定的に設定されるため，各時刻でどのタスクと Co-run するかが一意に定まり，Co-run の影響を制御することが可能となる．

3.2 スケジュールテーブル

TDMA スケジューリングを行う場合，スケジュールテーブルを実行前に決定する必要がある．本稿では Mixed Criticality System 向けのスケジュールテーブルを作成する．スケジュールテーブルの例を表 2 に示す．この例は表 1 のタスクセットの場合の例で，コア数は 2 コアを想定している．○は実行状態，空欄はアイドル状態を示す．スケジュールテーブルは CL 毎に作成される．ここで，**CL＝Xのスケジュールテーブルの定義は全タスクが$C(X)$で実行完了した場合のスケジュールテーブルである**．スケジュールテーブルの規則を以下に示す．

(1) スケジュールテーブルに登録されている全タスクがデッドラインを満たす．ただし，$L_i < X$に対応する実行時間は 0 とみなし，テーブルに登録しない．

(2) 1 つの CL のスケジュールテーブル内では同時刻でコア数以上の並列度で実行しない(ただし，全 CL のスケジュールテーブルを合計した並列度はコア数以上となってもよい．この場合のスケジューリング方法は 3.3 節で後述する)．

(3) $X2 > X1$のとき，CL= $X1$のスケジュールテーブルでタスクiが実行完了する以前は，同一周期内において，タスクiのスケジュールテーブルの内容は CL= $X1$と CL= $X2$で同一である．

(1)で$L_i < X$に対応する実行時間を 0 とする理由は，3.3 節で後述する実行時スケジューラの動作により，CL が高いタスクが優先的に動作することを保証しているからである．よって，CL＝X のスケジュールテーブルの定義より，(1), (2)を満たせば 2 章で述べた Mixed Criticality Scheduling のスケジューラブル判定基準を満たすことが可能である．(3)は異なる CL のスケジュールテーブル間での整合を取るための規則である．これについては，実行時スケジューラの動作と併せて説明する必要があるため，3.3 節で後述する．

表 2：Mixed Criticality System のスケジュールテーブル例

CL=H のスケジュールテーブル

	時刻				
	0	1	2	3	4
A	○	○	○	○	○
B	○	○		○	

CL=M のスケジュールテーブル

	時刻				
	0	1	2	3	4
A	○	○	○	○	
B	○	○			
C			○	○	○

CL=L のスケジュールテーブル

	時刻				
	0	1	2	3	4
A	○	○			
B	○				
C			○		
D		○	○	○	

Vol.2021-ARC-244 No.21
Vol.2021-SLDM-194 No.21
Vol.2021-EMB-56 No.21
2021/3/25

3.3 実行時スケジューラの動作

本節では，スケジューリングテーブルを実現するための実行時スケジューラの動作について説明する．実行時スケジューラは毎時刻以下の動作を行う．

- 全 CL のスケジュールテーブルの中で，その時刻に実行予定となっているタスクのうち，まだ実行完了していないタスクを実行状態にする．
- 条件を満たすタスクがコア数以上存在した場合，クリティカリティレベルの高いタスクを優先して実行状態にする．

表 2 を用いて具体例を説明する．時刻 0 ではスケジューラはタスク A と B を実行状態にする．時刻 1 でスケジューラに登録されているタスクは A，B，D である．このとき，タスク B は$C(L) = 1$なので，実行完了している可能性がある．もし B が実行完了していなければタスク A，B を実行状態にする．B が実行完了している場合は，時刻 1 でタスク A，D を実行状態にする．時刻 2 でスケジューラに登録されているタスクは A，C，D である．このとき，タスク A は$C(L) = 2$なので，実行完了している可能性がある．もし A が実行完了していなければタスク A，C を実行状態にする．実行完了している場合は，タスク C，D を実行状態にする．

ここで，3.2 節でスケジュールテーブルの規則(3)を設定した理由について実行時スケジューラの動作と併せて説明する．低い CL のスケジュールテーブルでタスクの実行完了予定時刻以前のある時刻で，2 つのスケジュールテーブルの間で実行状態が異なると，その時刻でそのタスクを実行すべきかどうかが判断できない．

図 1 に例を示す．CL=H(High)と CL=L(Low)の 2 つのクリティカリティレベルがあるとする．(a)のケースでは，タスク A は時刻tでは CL=L のスケジュールで実行完了していない．この時，時刻tで 2 つのスケジュールテーブル間で実行状態が違うため，タスクは実行すべきかどうか判断できない．時刻tでタスク A を実行しなければ，タスク A は CL=H のスケジュールテーブルのデッドラインを守れない．一方時刻tでタスク A を実行すれば，CL＝L で実行を予定していた別のタスクがその時刻で実行できなくなり，その別タスクが$C(L)$で実行完了したとしてもデッドラインミスを発生してしまう可能性がある．

一方，(b)のケースでは，タスク A は時刻tで既に CL=L のスケジュールで実行完了している．時刻tで 2 つのスケジュールテーブル間で実行状態が違うが，実行時のタスク A の進捗状態に応じて実行状態を決定できる．時刻tでタスク A がまだ実行完了していないと実行時に判断された場合，タスク A は$C(L)$を超過したということであり，実行時スケジューラは CL=H のスケジュールテーブルの通りタスク A を実行すればよいと判断できる．

以上のような理由から，3.2 節でスケジュールテーブルの規則(3)を設定した．

最後に，Mixed Criticality Scheduling のスケジューラブル判定基準の充足性について説明する．3.2 節で述べたように，スケジュールテーブルは Mixed Criticality Scheduling のスケジューラブル判定基準を満たすように作られる．そのため，ここでは，**全てのタスクが CL=X に対応する実行時間以内で完了した場合，CL=X のスケジュールテーブル通りに実行される**ことを説明する．CL が X2 である任意のタスク i があるとする．ある時刻 t で，CL=X2 と CL=X1(X2>X1 とする)の 2 つのスケジュールテーブル間で実行予定のタスクが異なり，かつ時刻 t でタスク i が実行完了していない場合，タスク i は X1 に対応する実行時間($C(X1)$)を超えて実行しているということである（なぜならば，3.2 節のスケジュール規則(3)より，タスクが$C(X1)$以内で完了すれば 2 つのスケジュールテーブルの内容はそのタスクに関して同様なため）．実行時スケジューラは高い CL のタスクを優先するため，$C(X2)$の実行時間を超えて実行する，CL が X2 より高いタスクが無い限り，CL=X2 のスケジュールテーブル通りに実行されることが保証される．

以上から，実行時スケジューラの動作により，全てのタスクが CL=X に対応する実行時間以内で完了した場合，CL=X のスケジュールテーブル通りに実行されることが保証される．

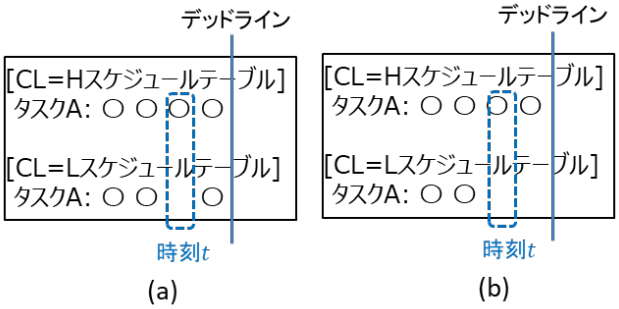


図 1：スケジュールテーブル間の整合・不整合ケース

4. スケジュール探索手法

本章では，3 章で説明した Mixed Criticality Scheduling のスケジューラブル判定基準を満たすスケジュールテーブルを探索する方法について説明する．

4.1 線形計画問題化によるスケジュール設計

マルチコアのスケジュール探索問題は NP 困難クラスに属する問題であることが知られている[8]．このような複雑な問題を求解する方法として，問題を線形計画問題化し，線形計画問題に対応した最適化ソルバを活用する方法がある．この方法により，最適化ソルバの高速な求解アルゴリズムを活用することが可能となる．そこで本稿では，スケジュールテーブル導出を線形計画問題に定式化することとした．

Vol.2021-ARC-244 No.21
Vol.2021-SLDM-194 No.21
Vol.2021-EMB-56 No.21
2021/3/25

4.2 Co-run モデル

Co-run による実行時間増加度合いは Co-run するタスクの組み合わせによって異なる．しかし，タスクの組合せはタスク数によって指数関数的に増大するため，すべての組合せにおける Co-run 影響度を事前に測定するのは現実的ではない．そこで，Co-run に関わる性質を以下のようにモデル化することとした．

- タスクは影響を与えやすいタスクと与えにくいタスクに二分する．前者をセンシティブタスク，後者を非センシティブタスクと呼ぶ．
- 非センシティブタスクとの Co-run による実行時間の増加影響は 0 と考える(1 つとの Co-run 時も複数との Co-run 時同様に 0).
- 1 つのセンシティブタスクとの Co-run による実行時間増加度合いは，影響を受けるタスク側の性質によって決まる．影響を受ける側のタスクごとに実行時間増加度合いが決まるが，Co-run した相手のタスクには依存せず一定量の実行時間が増加する．
- 複数のセンシティブタスクとの Co-run による実行時間増加度合いはセンシティブタスクとの Co-run 数に依存する．センシティブタスクとの Co-run 数が大きいほど実行時間増加度合いは大きくなる（なぜならば，Co-run するタスク数によって影響度合いが大きく変化することが報告されているため[1][2]）．
- Co-run による実行時間増加度合いは，Co-run する各タスク内の処理フェーズによらず一定である．
- タスクiがm個のセンシティブタスクと Co-run したときの実行時間増加度合いを$R_{i,m}$とする．$R_{i,m}$を Co-run 影響度と呼ぶ．$R_{i,m}$の算出式を以下に示す．

$$R_{i,m} = \frac{(C_{co,} - C_{single})}{C_{single}}$$

ただし，C_{co}は Co-run 時の実行時間，C_{single}は単独実行時の実行時間である

4.3 定式化

本節ではスケジューリングテーブル導出問題の線形計画問題への定式化について説明する．

4.3.1 リアルタイム制約

リアルタイム制約を定式化するため，タスクの実行状態を示す実行状態変数$Z_{i,l,t}$を導入する．図 2 にタスク実行状態変数の例とリアルタイム制約の関係を示す. 図 2 中の 0, 1 の数字は実行状態変数$Z_{i,l,t}$を示しており，1 が実行，0 が実行しないことを示す．$S_{i,k}$，$D_{i,k}$はそれぞれタスクiのk番目のジョブのリリースタイム，絶対デッドラインを示す．この例ではタスク A，B の 2 つが存在しており，簡単のためリリースタイム，絶対デッドラインはどちらも同じであるとしている．

リアルタイム制約として①,②の２つを設定した. 以降，各制約について説明する．

①：各周期内でコア占有時間が実行時間に等しい

$$\forall i, l, k, \quad \sum_{t=S_{i,k}}^{D_{i,k}} Z_{i,l,t} \geq \bar{C}_{i,l} + \sum_{m=1}^{N_c-1} (R_{i,m} \cdot \sum_{t=S_{i,k}}^{D_{i,k}} \varepsilon_{i,\,l,t,m}\ ) \quad (1)$$

各変数の意味は次の通りである．

i：タスク ID、 k：ジョブ ID、 l：クリティカリティレベル ID，$D_{i,k}$：タスクiのジョブkの絶対デッドライン，$C_{i,l}$：タスクiのクリティカリティレベルlに対応する実行時間、$\bar{C}_{i,l}$：タスクiのクリティカリティレベルlに対応する補正実行時間，N_c：コア数，$S_{i,k}$：タスクiのジョブkのリリースタイム，L_i：タスクiのクリティカリティレベル，$Z_{i,l,t}$：時刻tでのタスクiのクリティカリティレベルlにおける実行状態変数，$\varepsilon_{i,\,l,t,m}$：時刻tでタスクiと Co-run する可能性のあるセンシティブタスク数がmで，かつタスクiが時刻tで CL=lのスケジュールテーブルで実行するならば 1，それ以外は 0．ただし，$m = N_c - 1$ の時は，Co-run するセンシティブタスク数がコア数以上のときに 1，それ以外は 0.

本稿では$\varepsilon_{i,\,l,t,m}$を Co-run 状態変数と呼ぶこととする．Co-run 状態変数の詳しい性質については 4.3.5 項，線形式で表現する方法については 4.3.6 項で後述する．また，補正実行時間とは，タスクの CL より高い CL に対応する実行時間を 0 としたものである．

式(1)の左辺は CL=lスケジュールテーブル上での，各周期においてタスクiに割り当てられる CPU 時間の合計である．右辺第 1 項は補正実行時間であり，第 2 項は Co-run の影響による実行時間の伸びを示している．補正実行時間と Co-run の影響による実行時間の伸びの和よりも，各周期において割り当てられる CPU 時間が多ければデッドラインを満たすことができる．

第 2 項について詳細に説明する．$\varepsilon_{i,\,l,t,m}$の時刻tに関する和は，周期内でタスクiがm個のセンシティブタスクと Co-run する回数である. その結果と$R_{i,m}$の積を取り, Co-run 数mに関する総和を取ることで，Co-run による実行時間の伸びが求まる．

ただし，式(1)のように定式化した場合，Co-run により実行時間の伸びを過大評価してしまうという問題がある．表 3 に説明用の仮想タスクセットを示す．ここでは単一クリティカリティレベルであり，2 コアを仮定している．表 3 の Sen 列はセンシティブタスクか否かを表しており，両タスク共にセンシティブタスク(Yes)としている．スケジュールの期待値は表 4 の通り．タスク A は$R_{A,1}$=0 なので，タスク A は Co-run の影響は受けず, $C_{A,1} = 5$の通り CPU 実行時間が割り当てられている．タスク B は$R_{A,1}$=0.5 であり，Co-run の影響を受ける．時刻 0，1 で 2 回の Co-run が発生しており，Co-run の実行時間伸びは 0.5×2=1.0 である．よってタスク B には 3 の CPU 実行時間が割り当てられることが期待される．一方，式(1)の制約を課した場合に導出される

Vol.2021-ARC-244 No.21
Vol.2021-SLDM-194 No.21
Vol.2021-EMB-56 No.21
2021/3/25

スケジュールテーブルは表 5 の通り. タスク B の実行時間は 2 であるため，式(1)の右辺が 3 となる．この影響により周期内にタスク B に割り当てる CPU 時間(式(1)の左辺)が少なくとも 1 増える. $Z_{i,l,t}$が増えた影響で Co-run する回数，つまり$\sum_{t=S_{B,k}}^{D_{B,k}} \varepsilon_{B,l,t,m}$が 1 増え，式(1)の右辺第 2 項が 0.5 増加し，式(1)の制約を満たすために周期内にタスク B に割り当てる CPU 時間は 3.5 以上必要となる．増えた分も同様に Co-run の影響が増加するが，その影響の反映後の式(1)の右辺は 4 となるため，最終的に$Z_{i,l,t}$の総和は 4 に収束する．以上の理由から，表 5 のようにタスク B には CPU 時間 4 が割り当てられる．Co-run により実行時間の伸びによりさらに Co-run 時間が増えたとカウントしており, Co-run の影響を過大評価しているといえる．これはスケジューラビリティの低下につながるが，その影響度は 5.2 節で評価する．

表 3：Co-run 過大評価説明用タスクセット

タスク名	T	D	$\bar{C}$	R	Sen
A	5	5	5	0	Yes
B	5	5	2	0.5	Yes

表 4：スケジュールテーブルの期待値

	時刻				
	0	1	2	3	4
A	○	○	○	○	○
B	○	○	○		

表 5：式(1)から導出されるスケジュールテーブル

	時刻				
	0	1	2	3	4
A	○	○	○	○	○
B	○	○	○	○	

制約②：各周期内でデッドライン以上は全て 0

$$\forall i,l,k,\quad Z_{i,l,t}=0 \quad if\ D_{i,k} \le t < S_{i,k+1} \qquad (2)$$

デッドライン以降の時刻で$Z_{i,l,t}=1$となると，デッドラインミスを発生していることになる．よって式(2)の制約を課す．

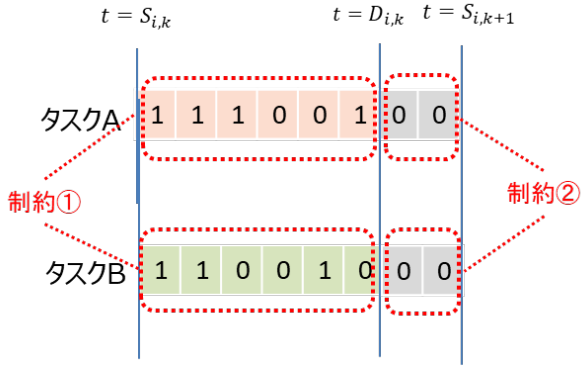


図 2：タスク実行状態変数$Z_{i,l,t}$とリアルタイム制約

4.3.2 コア数制約

3.2 節で述べたように，1 つの CL スケジュールテーブル内でコア数以上の並列度で実施してはいけない．

制約③：各時刻でコア数以上の並列度で実行しない

$$\forall t,l,\quad \sum_{i=1}^{N_T} z_{i,l,t} < N_c \qquad (3)$$

ただし，N_Tはタスク数である．

4.3.3 スケジュールテーブル間制約

各 CL のスケジュールテーブル間で 3.2 節の(3)で示した条件を満たす必要がある．タスクごとに，2 つの異なる CL（l_2，l_1）のスケジュールテーブル間の関係を考える．また，2 つの CL の関係は$l_2 > l_1$であるとする．タスクiが時刻t_1で CL=l_1のスケジュールテーブルで実行が完了してない状態とは，t_1以降に完了する場合か，t_1で丁度完了する場合である．

t_1以降に完了する場合，$Z_{i,l_2,t_1} \neq Z_{i,l_1,t_1}$となるパターンは図 3 の(a),(b)どちらかである．どちらの状態も以下の制約式を課すことで防ぐことができる．

$$(l_2 > l_1) \wedge (t_1 < t_2) \Rightarrow Z_{i,l_1,t_2} + Z_{i,l_2,t_1} - Z_{i,l_1,t_1} \le 1 \qquad (4)$$

一方，t_1で丁度完了する場合は図 3 の(c)のケースである．この時，以下の制約式を課すことで(c)の状態を防ぐことができる．

$$(l_2 > l_1) \wedge (l_2 \le L_i) \Rightarrow Z_{i,l_2,t} \ge Z_{i,l_1,t} \qquad (5)$$

式(5)にて，$l_2 \le L_i$を条件にするのは補正実行時間の定義に起因する．$l_2 > L_i$ならば補正実行時間は$\bar{C}_{i,l_2}=0$である．よって，式(1)より，すべての時刻において$Z_{i,l_2,t}=0$は制約条件を満たすことになる．この状態で式(5)の$Z_{i,l_2,t} \ge Z_{i,l_1,t}$が課されていると，$Z_{i,l_1,t}=0$となってしまう．よって，$l_2 > L_i$の場合は$Z_{i,l_2,t} \ge Z_{i,l_1,t}$を課さないこととした．

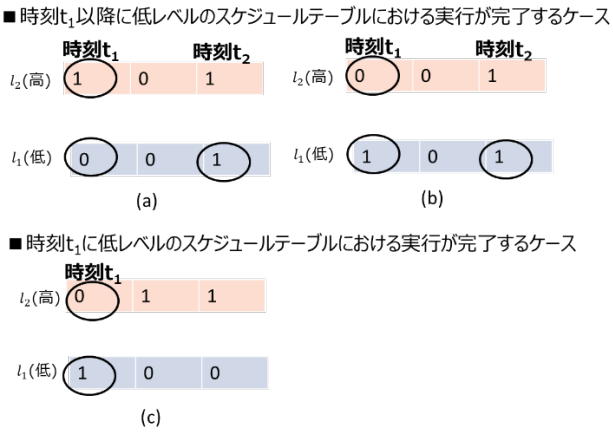


図 3：スケジュール間制約を満たさないケースの例

4.3.4 目的関数

1 章で述べたように，本稿の目的はスケジューラビリティの向上と Mixed Criticality Scheduling のスケジューラビリティ判定基準を満たすことである．制約条件を満たすスケジュールを探索することで，上記の目的は既に満たしている．

Vol.2021-ARC-244 No.21
Vol.2021-SLDM-194 No.21
Vol.2021-EMB-56 No.21
2021/3/25

本稿では, 副次的な目的として CPU アイドル時間の最大化を目的関数とする．CPU アイドル時間が増えれば消費電力の観点でよりよいスケジュールであるといえる．CPU アイドル時間の最大化は, タスク割り当てられる CPU 時間の最小化と同義である．よって式(6)のような目的関数を設定した．

$$F = \arg\min\left(\sum_{i,l,t} Z_{i,l,t}\right) \qquad (6)$$

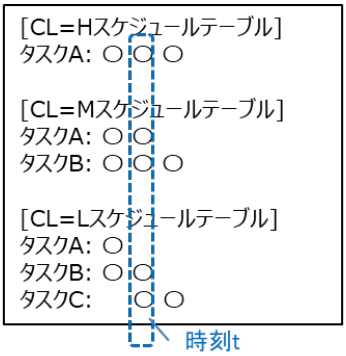
4.3.5 Co-run **状態変数の性質**

4.3.1 項で記載した Co-run 状態変数$\varepsilon_{i,\,l,t,m}$の定義を再掲する．

$\varepsilon_{i,\,l,t,m}$：時刻tでタスクiと Co-run する可能性のあるセンシティブタスク数がmで，かつタスクiが時刻tで CL=lのスケジュールテーブルで実行するならば 1，それ以外は 0．ただし，$m = N_c - 1$ の時は，Co-run するセンシティブタスク数がコア数以上のときに 1，それ以外は 0.

まず, Co-run する可能性のあるタスク数の意味を説明する．図 4 は 2 コアを想定したスケジュールテーブルの例である．$L_A = H, L_B = M, L_C = L$である．全スケジュールテーブルのどこかで，その時刻で実行する可能性のあるタスク数が Co-run する可能性のあるタスク数である．図 4 の場合タスク A，時刻t における Co-run する可能性のあるタスク数はタスク B,タスク C の 2 である（コア数以上の並列度で実行することはないため，実際に Co-run するタスク数は 1 である）．

次に, Co-run 状態変数の定義が$m = N_c - 1$ の時とそれ以外で異なる理由について説明する．全スケジュールテーブルを合計すると，ある時刻 t でコア数以上のタスクが実行予定となる可能性がある（3.2 節(2)参照）．ただし 3.3 節で説明した実行時スケジューラの動作により，実行時にコア数以上の並列度で実行されることはない．そこで Co-run する可能性のあるセンシティブタスク数が$m \geq N_c - 1$の時は, $N_c - 1$個のタスクと Co-run したと考える．よって，Co-run 状態変数の定義は$m > N_c - 1$の時，つまり Co-run する可能性のあるセンシティブタスク数がコア数以上のときに 1 となるようにしている．



$N_c = 2, L_A = H, L_B = M, L_C = L$

図 4：同時刻でコア数以上のタスクの実行が予定される例

4.3.6 Co-run **状態変数の線形化**

次に Co-run 状態変数$\varepsilon_{i,l,\,t,m}$の線形化について説明する．$\varepsilon_{i,l,\,t,m}$は最適化対象の変数(最適化変数)である．Co-run するセンシティブタスク数mは同じく最適化変数である$Z_{i,l,t}$によって定まる変数であるため, 上述の定義を満たすように, $\varepsilon_{i,l,\,t,m}$と$Z_{i,l,t}$の関係を線形制約式で表現する必要がある．$\varepsilon_{i,l,\,t,m}$は次式で表すことができる．

$m \neq N_c - 1$の時：

$$\varepsilon_{i,\,l,t,m} = \begin{cases} 1 \ (if \sum_{j \in J_s - i} Z_{j,L_j,t} = m \wedge Z_{i,l,t} = 1) \\ 0 \ (otherwise) \end{cases} \qquad (7-1)$$

$m = N_c - 1$の時：

$$\varepsilon_{i,\,l,t,m} = \begin{cases} 1 \ (if \sum_{j \in J_s - i} Z_{j,L_j,t} \geq m \wedge Z_{i,l,t} = 1) \\ 0 \ (otherwise) \end{cases} \qquad (7-2)$$

ただし，jは Co-run タスク ID，J_sはセンシティブタスクの集合である．$\sum_{j \in J_s - i} Z_{j,L_j,t}$は Co-run 数である．

まず，式(7-1)の線形化について説明する．式(7-1)の条件式を以下のように(A)とおく．

$$\left(\sum_{j \in J_s - i} Z_{j,L_j,t} = m \wedge Z_{i,l,t} = 1\right) \Rightarrow \varepsilon_{i,\,l,t,m} = 1 \qquad (A)$$

ここで最適化変数$\varphi_{m,i,t}$を導入する．

$\varphi_{m,i,t}$：時刻tでタスクiが Co-run するセンシティブタスク数がmであれば 1

すると，

(A)⇔ $m(1 - \varphi_{m,i,t}) \neq \sum_{j \in J_s - i} Z_{j,L_j,t}$ ・・・・・・・・・(B)

∧

$0 \leq \varphi_{m,i,t} + Z_{i,l,t} - 1.5\varepsilon_{i,\,l,t,m} < 2$・・・・・・・・(C)

ただし，J_s：センシティブタスクの集合である．(B)の右辺は時刻tでタスクiと Co-run するセンシティブタスク数である．(B) の右辺= mの場合$\varphi_{m,i,t}$=1 が保証される．右辺≠ mの場合$\varphi_{m,i,t}$は不定である．(B)の等号否定を 2 つの不等号で表すことで(B)は線形式となる．(C)も明らかに線形式である．以上で(7-1)を線形化できた．なお，(B)の右辺≠ mの場合$\varphi_{m,i,t}$は不定であることから，(A)の条件式を満たさない場合の$\varepsilon_{i,\,l,t,m}$の値も不定である．(A)の条件式を満たさない場合に$\varepsilon_{i,\,l,t,m}$＝0 とする条件式を設定する方法もあるが, 最適化変数増加による計算時間増加を回避するため，本稿では条件式を設定しないこととした．この妥当性は，本稿では目的関数を式(6)のように設定していることから説明できる．式(6)により，$Z_{i,l,t}$は小さくなるように抑えられるため，式(1)より$\varepsilon_{i,\,l,t,m}$も小さくなるように抑えられる．よって，不当に$\varepsilon_{i,\,l,t,m}$が大きくなることはない．

次に，式(7-2)の線形化について説明する．式(7-2)は$m = N_c - 1$の時に成立する式なので以下のように表すことができる．

$$\varepsilon_{i,l,t,N_c-1} = \begin{cases} 1 \ (if \sum_{j \in J_s - i} Z_{j,L_j,t} \geq N_c - 1 \wedge Z_{i,l,t} = 1) \\ 0 \ (otherwise) \end{cases} \qquad (7\text{-}2)'$$

式(7-2)’は以下の式(8-1)，式(8-2)と同値である．

Vol.2021-ARC-244 No.21
Vol.2021-SLDM-194 No.21
Vol.2021-EMB-56 No.21
2021/3/25

$$0 < N_c - 1 - \sum_{j \in J_s - i} Z_{j,L_j,t} + M\omega_{i,t} \leq M \qquad (8\text{-}1)$$

$$\wedge$$

$$0 \leq \omega_{i,l,t} + Z_{i,l,t} - 1.5\varepsilon_{i,l,t,N_c-1} < 2 \qquad (8\text{-}2)$$

ただし，$\omega_{i,l,t}$は 0 または 1 を取る最適化変数，Mは他項に比べて十分大きな数である．式(8-1)により以下を保証する．

$$\sum_{j \in J_s - i} Z_{j,L_j,t} \geq N_c - 1 \Rightarrow \omega_{i,t}=1 \qquad (8\text{-}1\text{-}1)$$

式(8-2)により以下を保証する．

$$\omega_{i,t} = 1 \wedge Z_{i,l,t} = 1 \Leftrightarrow \varepsilon_{i,l,t,N_c-1} = 1 \qquad (8\text{-}2\text{-}1)$$

式(8-1-1)，式(8-2-1)により，式(7-2)が成立する．式(8-1)，式(8-2)はそれぞれ線形式であり，以上で式(7-2)を線形化できた．

5. 評価

本章では，設計したスケジューリング手法の評価内容について説明する．評価は機能評価(5.1 節)と性能評価(5.2 節)を行った．評価環境を表 6 に示す．

表 6：評価環境

ソルバ	Gurobi Optimizer 9.0.0[9]
OS	Windows 10 Professional
CPU	Corei5-9500 3.0GHz，6 コア
メモリ容量	64GB

5.1 機能評価

機能評価は以下のクライテリアで実施した．

i. 各 CL テーブルでリリースタイムからデッドラインまでに，タスク単独の実行時間＋Co-run の影響による遅延以上の実行時間を確保すること
ii. 1 つの CL のスケジュールテーブル内では，全ての時刻において，コア数を超える並列度で実行しないこと
iii. Co-run の効果を正しく判定できていること，つまり Co-run 数に応じて実行時間が伸びていること
iv. $X2 > X1$のとき，各周期において，CL= $X1$のスケジュールテーブルでタスクiが実行完了する以前は，同一周期内において，タスクiのスケジュールテーブルの内容は CL= $X1$と CL= $X2$で同一であること

False Negative がないこと，すなわち，スケジューリング可能なものをスケジューリング不可と判定することがないことはクライテリアとしなかった．今回問題では，False Negative は安全側不具合である．そのため，False Negative が無いことは保証せず，その発生頻度について性能評価の中で評価することとした．

評価用のタスクセットは以下の条件で生成した．

- タスク数：区間[4,6]の整数をランダムに選択
- 実行時間：区間[1,周期/2] の実数をランダムに選択
- 周期：{4, 6, 12}からランダムに選択
- 終了時刻：12
- デッドライン：周期と同一
- コア数：{2,3}からランダムに選択
- CL 数：3
- 干渉影響度：区間[0,0.5] の実数をランダムに選択
- タスク数ごと、100 セットで検証

評価結果を表 7 に示す．本節冒頭で述べたクライテリアをすべて満たしているものを合格，1 つでも満たしていないものがあると不合格としている．表 7 の通り，不合格となった結果は無いことを確認した．

スケジューリング不可となったタスクセットの妥当性は検証されていないが，この観点は性能評価で評価するため問題ない．

表 7：機能評価結果

タスク数	合格	不合格	スケジューリング不可
4	87	0	13
5	72	0	28
6	60	0	40

5.2 性能評価

5.2.1 評価の観点

Co-run 時の干渉を可能な限り回避することによるスケジューラビリティ向上効果について，性能評価を行った． マルチコアにおける最悪実行時間は Co-run による性能劣化を考慮した値とすべきである．よって比較対象は，全ての時間で Co-run した場合をタスクの実行時間としてスケジューリングした場合とした．

スケジューラビリティの測定方法について，複数のタスクセットを生成してスケジュール探索し，スケジュール成功した割合を計測した．本稿でのスケジューラビリティの定義を以下に示す．

スケジューラビリティ[%]:スケジュール成功したタスクセット数÷全タスクセット数×100

5.2.2 評価パラメータ

タスクセットの CPU 使用率とスケジューラビリティは密接な関係がある．そこで CPU 使用率をパラメータとし，CPU 使用率ごとのスケジューラビリティを算出した．また，Mixed Criticality System を対象としているため，各タスクセットにおいて複数のスケジュールテーブルが存在するが，本稿では CPU 使用率を全 CL の平均値とした．また，コア数が多くなるほど，干渉回避の余地が多くなると考えコア数も評価パラメータとした．

各パラメータは次の通りである．

・2 コア：CPU 使用率 50%～100%の間で 10%間隔で測定
・4 コア：CPU 使用率 100%～400%の間で 20%間隔で測定

なお，各 CPU 使用率につき、100 個のタスクセットのスケジュールを計算した．

5.2.3 提案手法用のタスクセット生成方法

提案手法評価用タスクセットを次の方法で生成した．

Vol.2021-ARC-244 No.21
Vol.2021-SLDM-194 No.21
Vol.2021-EMB-56 No.21
2021/3/25

- 周期：{10,20,30,60}の中からランダムに決定
- デッドライン：周期と同値
- 実行時間：区間[1 以上, min(上位 CL の実行時間,周期)]の実数をランダムに決定
- タスクの CL：区間[1,3]の整数値をランダムに決定
- 干渉影響度：区間[0,0.5]の実数をランダムに決定。ただし、(N 個のタスクと Co-run 時の干渉影響度)<(N+1 個のタスクと Co-run 時の干渉影響度)
- センシティブタスク特性：センシティブ or 非センシティブをランダムに選択

加えて，タスクセットは次の条件を満たすようにしている．

タスクの CL が最大のタスク数≥コア数＋1

この理由は，タスク数がコア数以下の場合，必ずスケジュールできてしまうからである．最大 CL のスケジュールテーブルに登録されるタスク数が最も小さいため，タスクの CL＝最大 CL となるタスクの数に下限を設けた．

5.2.4 比較対象の評価方法

比較対象は提案手法用のタスクセットを元に，各タスクの実行時間を以下に示す方法で算出したものに入れ替えたものを入力タスクセットとした．

$$\hat{C}_{i,l} = C_{i,l} + C_{i,l} \times R_{i,N_c-1}$$

$\hat{C}_{i,l}$：タスク i,CL=l の Co-run 影響度最大の実行時間、

C_{i,L_i}：タスク i，CL=l の(単独)実行時間

比較対象のスケジューラビリティ探索時は，提案手法の制約式のうち式(1)を以下に修正したものを用いてスケジュール探索を行った．式(1)′は式(1)から Co-run による影響を除いたものである．その他は提案手法と同様の制約式を用いている．

$$\forall i,l,k, \quad \sum_{t=S_{i,k}}^{D_{i,k}} Z_{i,l,t} \geq \hat{C}_{i,l} \qquad (1)'$$

5.2.5 タイムアウト時間の設定

提案手法は Co-run の影響を考慮するために，最適化変数を追加している．そのため，比較対象に比べて探索時間が増加してしまう．そこで本評価では，スケジュール探索時のタイムアウトを設定し，タイムアウトしたものは現実的な時間で解を求めることができなかったと考え，スケジュール不可であるとみなした．本評価におけるタイムアウト時間を 2 時間に設定した．

5.2.6 その他諸条件

スケジュール求解対象のタイムスロット数は周期の最小公倍数である 60 までとした．また，スケジューラビリティを評価するため，解が 1 つ見つかれば以降の探索は不要である．よって，提案手法，比較対象どちらも目的関数は設定せずに探索を行った．

5.2.7 評価結果

2 コアの評価結果を図 5 に，4 コアの評価結果を図 6 にそれぞれ示す．2 コア，4 コアどちらも比較対象よりも高いスケジューラビリティを示すことが確認できた．4.3.1 項で述べたように，提案手法の定式化は Co-run の影響を過大評価する場合があった．また，5.2.5 項で述べたように，提案手法は比較対象に比べて探索時間が増加するために，タイムアウトによるスケジュール不可が増加する懸念があった．本評価結果から，上記のようなマイナス面の影響を提案手法による性能向上効果が上回ることがわかった．

また，表 8 にスケジューラビリティ改善度を示す．スケジューラビリティ改善度は比較対象と提案手法のスケジューラビリティの差分である．表 8 の結果から，2 コアよりも 4 コアの方が，スケジューラビリティ向上度合いが高いことを確認した．これは，コア数が増えるほど探索空間が大きくなり，Co-run 回避によりスケジュール成功する可能性が高まるからであると推察できる．

表 8：スケジューラビリティ改善度

	改善度(%)	
	2 コア	4 コア
中央値	4.0	19.0
平均値	4.3	20.8
最大値	14.0	44.1

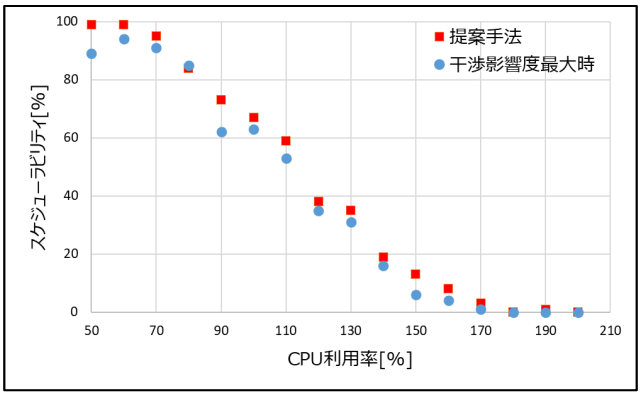


図 5：2 コア時のスケジューラビリティ評価結果

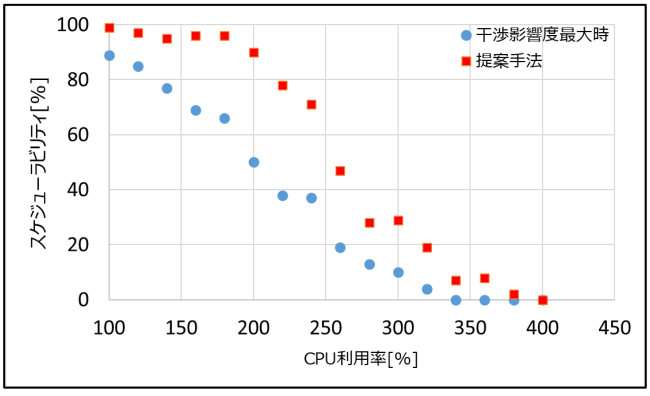


図 6：4 コア時のスケジューラビリティ評価結果

6. おわりに

本稿では，Co-run の影響による性能劣化を可能な限り回避しつつ，Mixed Criticality Scheduling のスケジュール判定基準を満たすことが可能なタスクスケジューリング手法を

Vol.2021-ARC-244 No.21
Vol.2021-SLDM-194 No.21
Vol.2021-EMB-56 No.21
2021/3/25

検討した．Co-run を制御するために，ベーススケジューリング手法として TDMA を採用し，TDMA で必要となるスケジュールテーブルの要件と，実行時スケジューラの動作を提案した．その上で検討した要件を満たすスケジュールテーブルを網羅的かつ効率よく探索するための手法を提案した．具体的にはスケジュール探索問題を線形計画問題に定式化し，線形計画ソルバを適用可能にした．

提案手法を用い，スケジューラビリティの改善効果を評価した．比較対象は Co-run の影響を最大に受けた場合を最悪実行時間とした上で，Co-run を考慮しない通常のスケジュール探索問題として求解した．その結果，2 コアの時に平均 4.3%，4 コアで 20.8%のスケジューラビリティ改善効果が得られた．

今後の課題として 2 点挙げられる．1 つ目は，提案手法における Co-run の影響過大評価の解消である．これにより，さらなるスケジューラビリティ向上が期待できる．2 つ目は，探索時間の削減である．提案手法では Co-run の影響回避のためスケジュール探索問題が複雑化した．その結果，各コア 1600 個のタスクセットに対し，2 コアでは 18 個，4 コアでは 71 個のタスクセットで 2 時間を超える探索時間を要した．一方，比較対象は探索時間が 2 時間を超えるケースは 0 であった．実用化に向けて，本課題を解決が必要であると考えている．

参考文献

[1] Yang Qin et al.:Analysis of Performance Degradation on Shared Cache in Multicore Systems. 2020, IPSJ SIG Technical Reports, vol 191, no38, p.1-7
[2] 婁宜之, et al.:マルチプロセッサにおける並列実行プログラムの共有キャッシュによる実行時間への影響度の見積もり手法. 情報処理学会研究報告, vol 50, no19, p1-8
[3] “Raspberry Pi 3 Model B+.” https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/
[4] VESTAL, S.:Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In Proceedings of the Real-Time Systems Symposium (Tucson, AZ, December 2007), IEEE Computer Society Press, pp. 239–243.
[5] M. Mollison, J. Erickson, J. Anderson, S. K. Baruah, and J. Scoredos: Mixed criticality real-time scheduling for multicore systems. 2010, In Proc. 7th IEEE International Conference on Embedded Software and Systems. 1864–1871
[6] S. Saidi, S. Steinhorst, A. Hamann, D. Ziegenbein and M.Wolf.: Special Session: Future Automotive Systems De-sign: Research Challenges and Opportunities. 2018, 2018 In-ternational Conference on Hardware/Software Codesignand System Synthesis (CODES+ISSS)
[7] J. Ren and L. T. X. Phan. : Mixed-criticality scheduling on multiprocessors using task grouping. 2015. In Proc. 27th ECRTS.IEEE, 25–36.
[8] J.A. Hoogeveen *et al*.: Complexity of scheduling multiprocessor tasks. with prespecified processor allocations. Discrete Applied Mathematics 55 (1994) 259-272
[9] Gurobi 製品紹介サイト, https://www.octobersky.jp/products/gurobi.html <accessed 2021-2-1>