# マルチコアアビオニクスソフトウェアにおけるデータ結合および制御結合の最適化：Xilinx Ultrascale+プロセッサでの事例


Alex Lim
LDRA Technology, Inc
Lewisville, TX, USA
alex.lim@ldra-usa.com

Roshini Ashok, Ph.D.
AvMC MCP-AI-Lab
DEVCOM AvMC
Huntsville, AL, USA

Ehsan Salehi
LDRA Technology, Inc
Lewisville, TX, USA
esalehi@lynx.com

Stephen DiCamillo
LDRA Technology, Inc
Lewisville, TX, USA
stephen.dicamillo@ldra-usa.com

William Vance
TriVector Services, Inc.
Huntsville, AL, USA

John Ross
AvMC MCP-AI-Lab
DEVCOM AvMC
Huntsville, AL, USA

Jay Thomas
LDRA Technology, Inc
Lewisville, TX, USA
jay.thomas@ldra-usa.com

Hugh Turner
Brockwell Technologies, Inc.
Huntsville, AL, USA



***概要*— 本論文では, セーフティクリティカルなアビオニクスソフトウェアでのマルチコア干渉の影響について, AMD Xilinx Zynq Ultrascale+ MPSoC プロセッサ上のリアルタイム OS (RTOS) LynxOSで動作させて調査する. 事例により, 実行予測可能性, システム信頼性, 最悪実行時間(WCET)に焦点を当てて, データ結合と制御結合(DCCC)を解析する. さらに, 静的解析でデータ結合や制御結合の解析を行うことによって, 詳細なデータ依存関係を明らかにし, 動的解析では, データサイズと干渉注入のさまざまなレベルを組合わせてWCETを測定する. 本論文では, 安全規格(AC 20-193[1], DO-178C[2])のオブジェクティブを満たすための指針を提供し, キャッシュの干渉に対する実験的な緩和策を提示する. また, 大きなデータバッファリングがシステム干渉に与える影響を検討し, 高信頼で認証可能なマルチコアアビオニクスシステムを開発するための実践的な知見を提供する.**



***キーワード*— *マルチコアプロセッサ, AC 20-193, データ結合, 制御結合, DO-178C, XilinxUltrascale***


## I. はじめに

セーフティクリティカルな組込みシステムにおいて, 複数コア間でのソフトウェアの相互作用を理解し, 管理することがますます重要になっている. 性能や電力効率に対する要求を満たすために組込み業界が複雑なマルチコアアーキテクチャへと移行する中で, ソフトウェアの正しさとトレーサビリティを確保することは, 特にアビオニクスや自動車, 防衛システムなどの分野で大きな課題となる. このような環境においては, 複数のソフトウェアコンポーネントが実行中にどのように相互作用し, 影響し合うか理解するためにデータ結合と制御結合を突き止めることが不可欠であり, それが難しく重要な課題となっている[3][4][5].

マルチコアシステムにおいては, データ結合の定義がコア間やメモリ階層間のデータの移動や処理にも拡張される. 一方, 制御結合の定義は, タスクや関数間の相互依存関係を表すものであり, 実行タイミングや同期に重大な影響を与える.

本論文では, AMD Xilinx Zynq Ultrascale+ MPSoC プロセッサとリアルタイムOS (RTOS) LynxOSを用いたセーフティクリティカルなアビオニクスソフトウェアにおけるマルチコア干渉の影響とデータ結合・制御結合解析の関係に重点を置いた詳しい事例を調査し, データサイズの違いが実行予測可能性やシステム信頼性, 最悪実行時間 (WCET) などの重要な指標にどのように影響するかという観点で検証する.

この研究では, 実例を用いて, セーフティクリティカルなアビオニクスシステムにマルチコア干渉が与える影響を評価する. Xilinxプロセッサはハードウェアプラットフォームの典型となるもので, 通信プロトコルやタスク同期が極めて重要な現実世界のシナリオをシミュレートする. 使用するLDRAツールスイート[6]は, システム内のデータ結合と制御結合を解析するために用いるDO-330[7]準拠のソフトウェア検証ツールで, テストケースの作成やタイミング情報の取得, さらに取得した情報の分析にも使用される. 干渉解析のためのソフトウェアアプリケーションをホストするよう, Lynx RTOSをさまざまに構成した.

以降, マルチコア干渉がソフトウェアシステムの静的・動的な特性にどのように影響するか調査する事例を説明する. 静的解析の観点で見ると, 重要なデータ結合を詳細にフィルタリングすることで, 結合性や複雑度が高い領域が明確になる. これは, タイミング違反につながる可能性のあるボトルネックを特定するのに役立つ. 動的解析の観点では, 本研究は, 独立したOSインスタンスが別々のコア上で動作し, それぞれのタスクを持つ場合に, タスクスケジューリング, データバッファリング, そして並列処理の影響を考慮してWCETを評価する. この二元的なアプローチにより, 干渉とタイミング結合の相互作用, そしてデータ結合・制御結合がどのようにして隠れた干渉を引き起こすかを包括的に評価できる. ここで, タイミング結合とは異なるコアでタスクや関数の並行実行がオーバーラップするタイプの結合である.

この調査結果は, AC 20-193 (AMC 20-193[8]), DO-178C (ED-12C[9]), 陸軍軍用耐空証明基準 (AMACC[10]) などの安




Reprinted with permission from the proceedings of DASC 2025.

全規格や耐空性規格において, マルチコア干渉がある場合のセーフティクリティカルなシステムに対する厳格な要件に対処することの重要性を強調している. AC 20-193 の勧告通達文書では, キャッシュ競合と共有リソース干渉が非決定性の主要因であると指摘し, 干渉の特定と緩和策の必要性を強調している.

干渉緩和の技法とそれがデータ結合・制御結合に与える影響を詳細に評価することで, 本論文は航空宇宙分野でマルチコアシステムを開発する技術者や研究者にとって実用的な指針となる. 調査結果は, 予測可能性, 信頼性, そして認証への即応を向上させるマルチコアシステムの最適化に対する知見を提供するものである.

## II. ハードウェアアーキテクチャと実験の設定

実験にはクアッドコアの ARM Cortex-A53 アプリケーションプロセッサ, デュアルコアの ARM Cortex-R5 リアルタイムプロセッサ, そして XCZU7EV-2FFVC1156 FPGA を搭載する Xilinx Zynq UltraScale+ ZCU106 評価ボードキット (図 1) を使用した. このボード上でリアルタイム OS の LynxOS を動作させ, 非対称マルチコア処理(AMP)でシステムを構成する. すなわち, 各プロセッサコアが自身に固有のオペレーティングシステムインスタンスで独立して動作する.

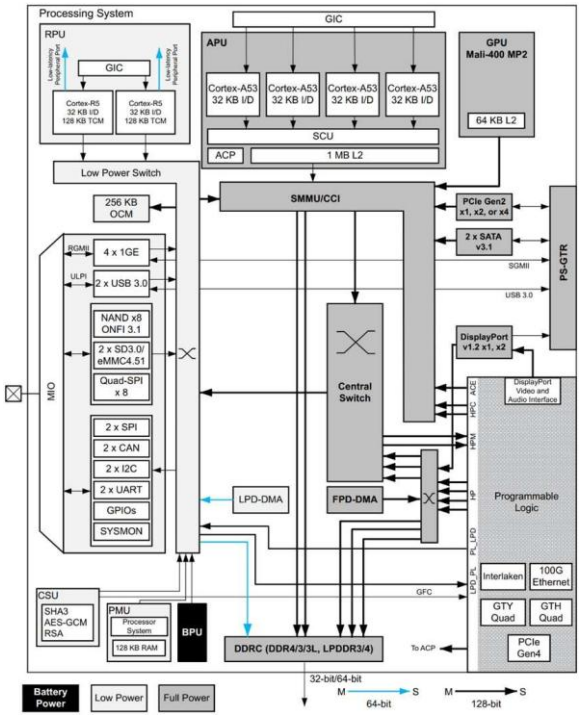


図 1. Xilinx Zynq UltraScale+ ZCU-106 評価ボード

複数のミッションがサポートできる実世界環境を表現し, 各 LynxOS インスタンスにハードウェアリソースを分離することで追加の共有ハードウェアリソースを減らすため, AMP を選んでいる.

特定のコアに静的に割当てられる分離パーティションを定義できるため LynxOS-178[11] を実験対象とした. この構成ではコアごとに L1 キャッシュが独立しており, 他の L1 キャッシュすべてとコヒーレンシはない. 実験では ARM Cortex-A53 コアのみを使用し, R5 コアはアイドルのままである. LDRA ツールスイートを用いて, プログラムをインスツルメントし, コードカバレッジ解析とデータ結合・制御結合解析, そして WCET 解析を行った結果について, 論文の後半で詳しく説明する.

事例は, 典型的なマルチコアプラットフォーム環境でのメモリ集約型アプリケーションである. ボード上の利用可能な共有リソース, すなわちメモリ, イーサネット, クアッドシリアル周辺インターフェース(SPI), IC 間通信(I2C), ダイレクトメモリアクセス(DMA) を評価する. 中でもメモリは, ほぼすべてのアプリケーションにわたり共通して依存するものであるために選ばれ, 主に L2 キャッシュを調査した. これは 4 つの A53 コアすべての共有リソースであるためである. メモリをマルチコア干渉のターゲットとすることは, データ結合と制御結合の解析において単一コア構成の場合よりも困難で現実的な例を提供する.

### A. ソフトウェアアプリケーション

ここではアプリケーションを「赤」と「青」で区別する. 青アプリは評価対象のベースラインアプリケーションを表し, 赤アプリは攻撃アプリケーションとして機能し, 高頻度の使用によって評価対象の共有リソースと干渉チャネルに負荷をかけるよう調整される[12][13]. メインアプリケーションである青アプリは, まず関数 rand() で得られるランダムな値を要素とする大きな配列を設定し, その要素が同サイズの別の配列のインデックスとなるようにする. インデックスは配列サイズに合わせて制約される. 次に, 2 つ目の関数がランダムなインデックス値によって最初の配列を使い, 2 つ目の配列をアクセスして, その配列の要素に特定の値を設定する. 青アプリは最初の LynxOS-178 インスタンスで動作した.

赤アプリも青アプリと同じことを行い, 配列のサイズも変動するが, アプリ内で配列サイズは同じである. ある配列のサイズが変更されると, もう一方の配列でも変更される. 赤アプリは別の A53 コアで独立して動作する 2 つ目の LynxOS-178 インスタンスを使用した. 青アプリと赤アプリはデータを共有せず, 意図されたデータ結合や制御結合もない.

### B. テストフレームワーク

データ結合と制御結合に関する静的解析には LDRA ツールスイートを用いた. このツールスイートはテストシナリオの作成や, テスト対象ソフトウェアアプリケーションの動的なタイミングデータの取得にも使用している. 図 2 は実験での高レベルの操作フローを示しており, 図 3 は実験で使用した開発/テストツールと成果物のブロック図を示している. 図 3 でオレンジ色のブロックは ZCU106 評価ボードのターゲットハードウェアである.

### C. テストシナリオ

各テストシナリオには 5000 回の繰返しがある. 青アプリで配列サイズの範囲は 24KB, 256KB, 512KB, 768KB, 1MB, 2MB であり, 赤アプリでも 24KB, 256KB, 512KB, 768KB, 1MB, 2MB であった. ベースラインテストは青アプリだけが動作するものであり, 青アプリの配列サイズと赤アプリの配列サイズのすべての組合わせをカバーするため, 合計 38 個の異なるテストシナリオを実施した.

### D. 検証テストケース

テストシナリオとは異なり, システムの挙動の検証と認証オブジェクティブの達成を目的とした 4 つのテストケースを秩序立てて定義している.

テストケース1(TCI_001)はXilinx Zynq UltraScale+ ZCU106評価基板キットの回路図を手動で検査するもので，干渉を引起こす可能性のある共有ハードウェアリソースを探す．このシナリオではクアッドコア間のL2キャッシュを共有リソースとして注目した．AMPの構成設定によって，最悪ケースを含むさまざまなシナリオでアプリケーションを実行できるよう分析と設計を行った．

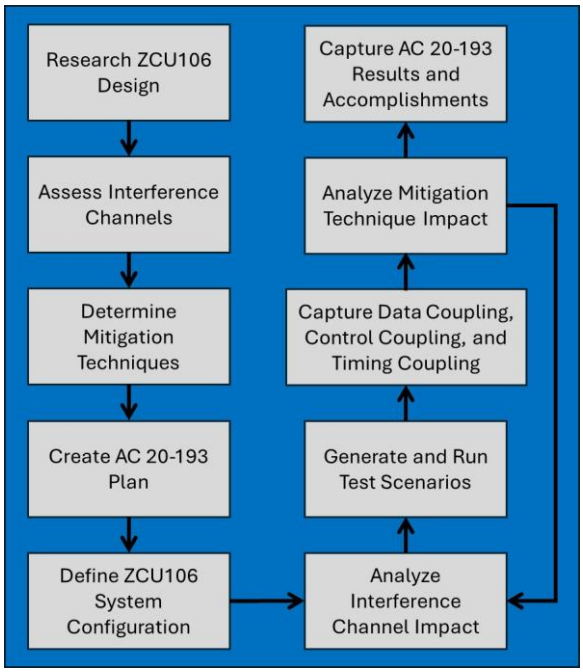



図2. 実験のフロー図

テストケース2(TCI_002)は，2つの独立したLynxOS-178インスタンスと第1コア上のハイパーバイザーをもつLynxSecure向けのLynxビルドの構成ファイルを手動で検査するものである．LynxOSとハイパーバイザー，1つめのLynxOS-178，2つめのLynxOS-178のハードウェアリソースを構成設定するスクリプトがある．この構成によって，L2キャッシュやメモリへのバスインターコネクトなど，ハードウェアアーキテクチャでは分離できないハードウェアリソースを除いて，LynxOS-178インスタンスそれぞれにハードウェアリソースを分離している．

テストケース3(TCI_003)では，上記のテストシナリオで示されたデータサイズが異なる38のシナリオで複数回の実験設定を実行する．青アプリと赤アプリでデータサイズの違いを組合わせて実行している．組合わせは，赤アプリがないものから，青アプリの配列サイズごとに異なるサイズの赤アプリを合わせるものまで多岐にわたる．それぞれの組合わせは，データのばらつきを考慮して5000回実行している．収集したデータにより，データサイズがL1を超えL2キャッシュ内に多く存在するほど，データサイズが大きくなるにつれて実行時間と干渉時間が増加すると予想される．青アプリと赤アプリ両方がL2キャッシュを使用する場合には，より多くの干渉が見られるはずである．

テストケース4(TCI_004)は，LDRAツールスイートが生成するデータ結合と制御結合レポートの解析である．データ結合レポートは，タイミング解析される関数が使用する大規模なデータ配列を明らかにする．制御結合とタイミング結合レポートは，これらの関数が並行実行されており，データサイズがL2キャッシュで使用するのに十分な大きさである場合にキャッシュ干渉を引起こしていることを示している．タイミング結合とデータ結合/制御結合については，最悪ケースを実行するよう実行シナリオを更新し，反復できる．

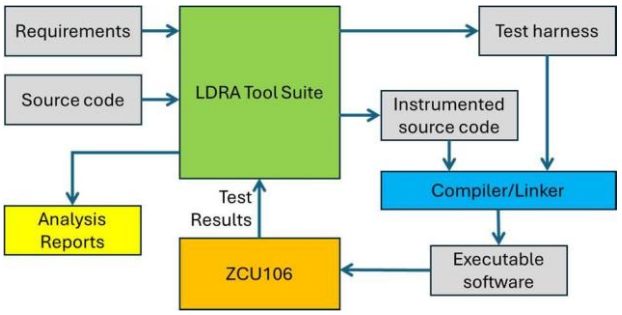



図3. 実験装置構成のブロック図

## III. マルチコアリアルタイムシステムにおける干渉解析とテスト技法

### *A. マルチコアリアルタイムシステムにおける干渉の定義と範囲*

マルチコア組込みシステムにおいて干渉とは，並行実行されるソフトウェアコンポーネント間のリソース競合によって引起こされる，意図しないタスク実行の遅延や妨害を指す．シングルコア環境では，逐次実行でタイミングの変動が主に制御フローの複雑さに起因するが，それとは異なりマルチコアプラットフォームでは共有のハードウェアリソースが導入されるので，それらがタスクのタイミング挙動に予測不能な影響を与える可能性がある[14][15].

この競合はレイテンシの増加や実行ジッタ，タイミングのデッドライン違反を引起こす可能性があり，安全性のため予測可能性と決定性が要求されるハードリアルタイムシステムにおいて，それらはすべて重大な懸念事項である．たとえば，あるコアで実行中の低優先度タスクが，キャッシュの追出しやメモリバスの飽和によって意図せずに別のコア上の高重要度のタスクを遅延させる可能性がある[16].

### *B. セーフティクリティカル分野での関連性*

セーフティクリティカルなシステム，特にDO-178CやISO 26262[17], IEC 61508[18]の対象となるシステムにおいては，このような意図しないタイミング変動は，特定されて制限され，そして緩和されなければ，許容されない．DO-178Cガイドラインにはマルチコアシステムにおける干渉に関して曖昧な点があり，その問題はAC 20-193で対処されている．本論文ではAC 20-193で概説されたオブジェクティブを明確にし，達成することを目的とする．それにはマルチコアソフトウェアコンポーネント間の干渉によって決定性や機能安全性が損なわれないという証拠が必要となる[1][14][15].

### *C. 干渉解析の範囲*

識別・検出・緩和・検証のプロセスは，選択されたターゲットハードウェアに完全に依存するものである．干渉解析は，選択したターゲットハードウェアによって完全に変わることがあり，その可能性が高い．干渉解析の範囲は，共有リソース使用状況の特定にとどまらず，次を含む．

- 検出：リソース競合がいつどこで発生しうるか特定する

- 定量化:干渉がタイミング挙動に与える影響を測定する
- 緩和：ソフトウェアアーキテクチャ(データサイズ削減やキャッシュ分割など)とハードウェア構成(キャッシュ分割や周辺機器の分離)を設計し, 共有リソースへのアクセスを分離またはスケジューリングする
- 検証：ツールやテスト技法を用いて, システムがあらゆる動作条件下で決定的に動作することを証明する

### *D. ZCU106における干渉*

TCI_001を通じて, 図4に示すようにZCU106上で複数の干渉チャネルが特定された. 干渉を引起こす可能性のある共有ハードウェアリソースは多数あるが, ここではアプリケーション処理ユニット(APU)に焦点を当てている. 具体的には, 独立したRTOSインスタンス上の独立したアプリケーションによるメモリ使用に起因する干渉について, L2キャッシュを調査している.

共有L2キャッシュによる干渉が発生し得る実際の航空電子機器環境をシミュレートできるよう, 特定の構成に合わせてハードウェアとRTOSを設定した後, それらの環境で動作するアプリケーションとして青アプリと赤アプリを開発した. 共有L2キャッシュを使用することによる潜在的な干渉を捕捉するため, TCI_003に対していろいろなテストシナリオを実行している.

最後に, 干渉が発生するタイミングと原因を特定するため, データ結合･制御結合解析をタイミング結合解析と組合わせて実施した. この解析の詳細を次節で説明する.

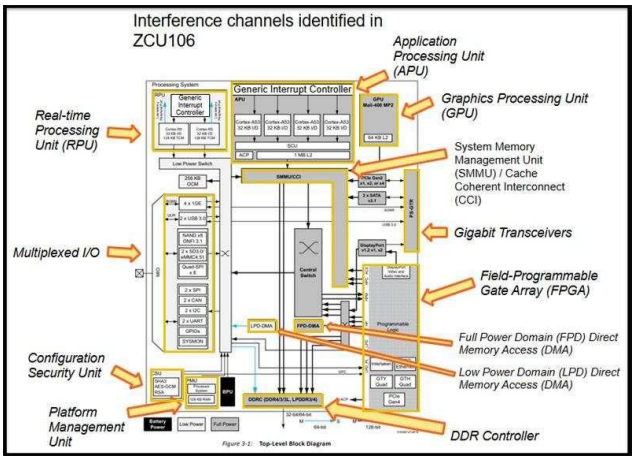



図4. ZCU106での干渉チャネル

## IV. マルチコア組込みリアルタイムシステムにおけるデータ結合･制御結合解析

データ結合と制御結合の解析は, 組込みシステムにおいて機能的な正しさを検証し, 安全規格への準拠を確保し, トレーサビリティを実現するために不可欠な手法である. データフロー解析はデータがどのように定義･使用され, 伝播するかを追跡し, 制御フロー解析は実行パスと論理の判断地点を検証する[3][4][5]. これらの手法を組合わせることで, 特にDO-178Cのような規格の対象となるソフトウェアにおいて, データ結合と制御結合を明らかにするための基盤が形成される.

アビオニクス, 自動車, 産業システムなどのセーフティクリティカルな分野では, 網羅的な静的および動的解析機能が不可欠である. データ結合と制御結合(DCCC)レポート, 制御フローグラフ(CFG), そしてデータフローパスは, DO-178CやISO 26262, IEC 61508[6]を含む大部分の機能安全規格への準拠を検証する場合の支援となる[6].

### *A. 静的解析機能*

静的解析器は, 実行前にソースコードを検査し, データ/制御の潜在的な異常を検出する. これによって次が可能になる.

- データの伝播を追跡する定義―使用の解析
- 未初期化変数やデッドコード, 到達不能な分岐の検出
- 制御フローグラフ(CFG), および論理と依存を表すデータフローパスの生成

マルチコア環境では, トレーサビリティと予測可能性を確保するためにソフトウェア構造がアーキテクチャ分割と整合しなければならず, これらの機能が特に重要になる[19].

### *B. 動的解析と要件トレーサビリティ*

通常, 動的解析には実行時の挙動を計測するため, コードへのインスツルメントが必要となり, それによってデータと制御の相互作用をリアルタイムで監視できる. マルチコアシステムでは, 動的解析を用いて次を特定し, 検証できる.

- 複数のコアやパーティションに跨る実行パス
- プロセス間通信とコア間での関数呼出し
- タスクの実行状況を解析するためのタイミング計測

### *C. タイミング結合*

独立したアプリケーションそれぞれのデータ結合･制御結合の情報と, 青アプリ, 赤アプリ両方からのタイミングデータとを組合わせることで[12][13][20], 干渉が発生するタイミングとその原因を分析できる. より具体的には, データ結合によって, どれかの関数においてL2キャッシュサイズに比べて大きなデータサイズが使用されることが明らかになる. また, ある関数に対して制御結合解析すると, その関数に結合している他の関数群がわかる. これによって, L2キャッシュを占有する大きなデータを使用する制御フローが明らかになる. この情報を別コア上で動作するアプリケーションからの結合データ, および両アプリケーションのタイミングデータと組合わせることで, 干渉が発生して実行中のタスクの実行時間が遅延する原因が分析できる.

独立したコア上の独立したRTOSインスタンスで動作する独立したアプリケーションのタスクや関数それぞれのタイミングデータを組合わせることで, タイミング結合解析の基盤が提供される. 従来, このタイミングデータは独立したアプリケーション1つだけで使用され, 複数のアプリケーション間には適用されていなかった. タイミング結合は, 複数コア間で干渉が発生している箇所とその原因を明らかにし, 以下に述べる2種類の解析を追加することでタイミング結合解析を補完できる.

第一の解析では, テスト対象の重要アプリケーション(青アプリ)の典型的な実行において, 他の独立したソフトウェア(赤アプリ)を実行せずに, 静的解析と動的解析情報を取得できる. これにはデータ結合, 制御結合とタイミングデータが含まれる. その後, 別のコア上で, 赤アプリを並行実行しながら同じシナリオを繰返す. 取得したデータを解析して干渉の有無を確認し, 結合データを調査することで, その原因が解明できる.

このデータにより,L2キャッシュが使用されるであろう大きなデータサイズを用いる関数群(関数グループA)が明らかになる. 次に制御結合により,関数グループAと結合する関数群が明らかになる. タイミング結合は,異なる2つのコアで並行実行される関数の実行時間を示している. この解析により,異なるコアで実行される独立した関数の中で,実行がオーバーラップして大きなデータサイズが使用されることでL2キャッシュ干渉を引起こしているものが明らかになる.

第二の解析では, 取得したデータ結合と制御結合の情報によって起こりうる最悪シナリオを特定する. 青アプリでもっと大きなサイズのデータと,赤アプリでもっと大きなサイズのデータを並行して用いることで, 各アプリケーションの制御フローに最悪ケースを実装して最悪の実行タイミングが捕捉できる. セーフティかつタイミングクリティカルなアプリケーションでは, その閾値を要件と照合することで, その要件が満たされているかを確認できる.

関数やタスク間のタイミング結合は, 独立したコア上で動作する独立したアプリケーション間で, 予期しない制御結合とデータ結合を結びつける. これにより,L2キャッシュのような共有リソースに起因する潜在的な干渉が発生する箇所を特定する分析が可能になる.

## V. 結果

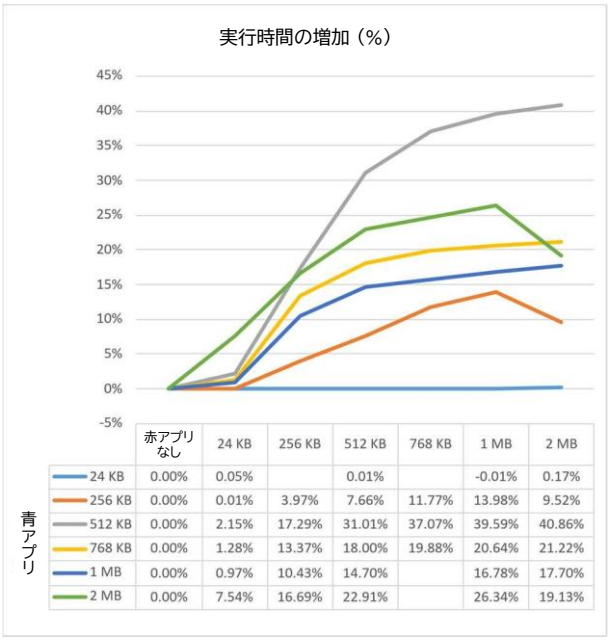
表 I に TCI_003 で実施した 38 個のテストシナリオにおける平均実行時間をナノ秒単位で示す. 青アプリのデータサイズが増加するにつれて実行時間が増加することが確認でき,赤アプリのデータサイズが増加する場合にも実行時間が増加することが確認できる. ただし, データサイズが十分小さくL1キャッシュだけを使用する場合を除く.

配列サイズがL1キャッシュサイズ(32KB)より小さい場合, 干渉は観測されず,ばらつきは許容誤差範囲内であった. アプリケーションがそれぞれ自身のコア上の独立したRTOSインスタンスで動作している場合でも, これは予想されるL2キャッシュ干渉と一致する.

予想通りの結果が得られている(図5). 最小サイズである24 KBでは, 赤アプリのサイズを大きくしても干渉は発生しなかった. しかし, L2 キャッシュに収まらない他のシナリオでは, 配列サイズが大きくなるにつれて干渉も増加した. 赤アプリが大きな配列サイズで実行されている場合に, 青アプリの上から三番目の関数の実行時間は約2%から最大約41%増加することが確認された. セキュリティクリティカルやタイミングクリティカルなアプリケーションでは,干渉によってこれほど実行タイミングが増加することは許容できない. 対象となるクリティカルなタスクは, 隔離された状態で独立して実行されるため,ソフトウェア開発者やテスターは,単にタスクの制御結合やデータ結合の観点から見るだけでは実行時間の増加に気づかないだろう.

図 6 は, 実験でのデータ結合と制御結合のレポートである. ここで data_creation_simulation 関数は青アプリで, また random_array_simulation 関数は赤アプリで動作している.

以下のいくつかの図表によって, この実験の重要な側面を示す. 図 7 に, データサイズが大きい int 型配列のグローバル変数 array_jump_val_rand[256000] が示されている. 図 8 は, この変数 array_jump_val_rand と関数 data_creation_simulation (double *)との結合を示しており, アプリ間で干渉が生じていることがわかる. この図で右側の制御結合レポートを見ると, main 関数が data_creation_simulation 関数を呼出していることがわかる.




| 青アプリ | 赤アプリなし | 24 KB | 256 KB | 512 KB | 768 KB | 1 MB | 2 MB |
|---|---|---|---|---|---|---|---|
| 24 KB | 0.00% | 0.05% | | 0.01% | | -0.01% | 0.17% |
| 256 KB | 0.00% | 0.01% | 3.97% | 7.66% | 11.77% | 13.98% | 9.52% |
| 512 KB | 0.00% | 2.15% | 17.29% | 31.01% | 37.07% | 39.59% | 40.86% |
| 768 KB | 0.00% | 1.28% | 13.37% | 18.00% | 19.88% | 20.64% | 21.22% |
| 1 MB | 0.00% | 0.97% | 10.43% | 14.70% | | 16.78% | 17.70% |
| 2 MB | 0.00% | 7.54% | 16.69% | 22.91% | | 26.34% | 19.13% |

図 5. 干渉による実行時間の増加

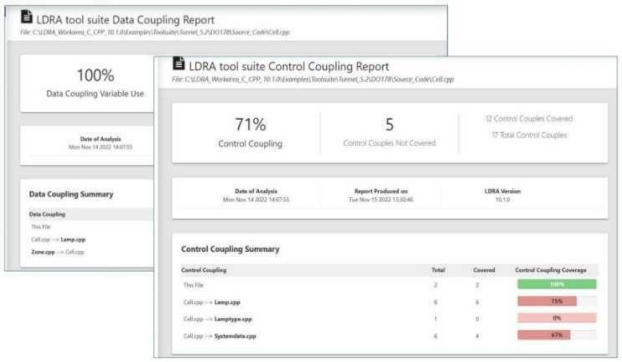



図 6. データ結合と制御結合のレポート

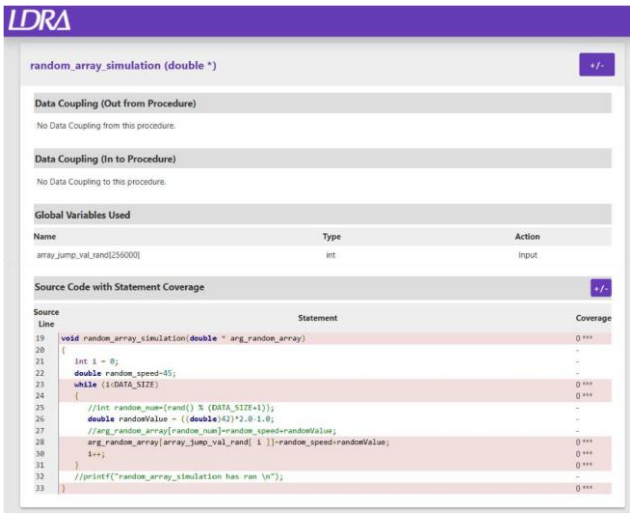



図7. 大きなデータサイズのグローバル変数

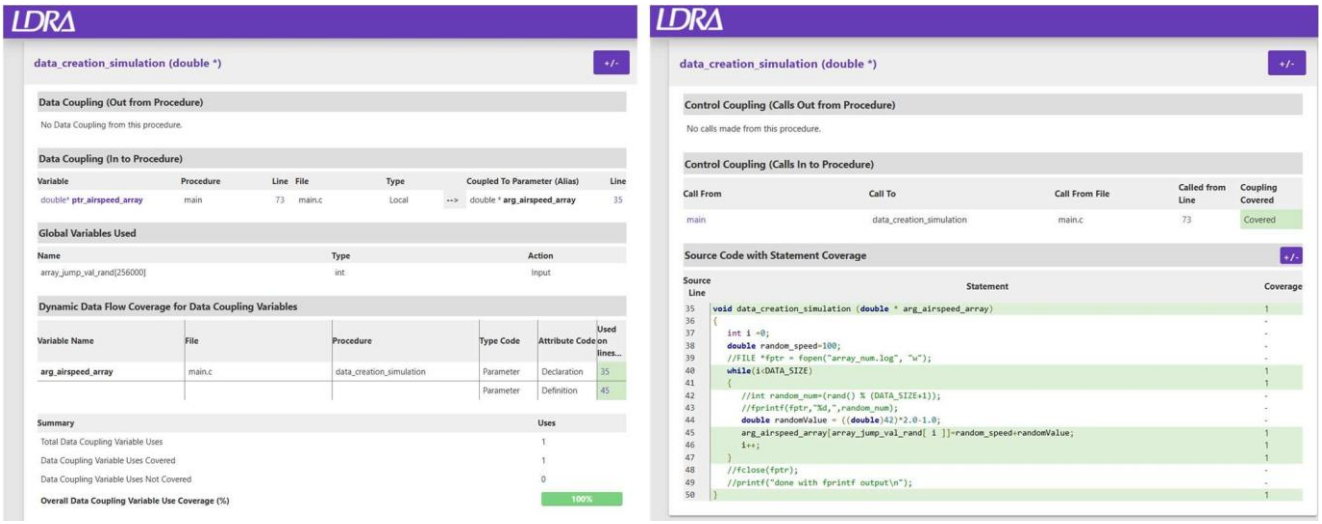



図 8. 大きなデータサイズのグローバル変数

data_creation_simulation と random_array_simulation 両関数間のタイミング結合の一部を表 II に示す. この表で関数 1 と関数 2 の列は, 並行動作しているこれら 2 つの関数を示す. 関数 1 が最初に始まって関数 2 は関数 1 の後に起動するが, それでも関数 1 と実行がオーバーラップする. 図 7 のデータ結合レポートに戻ると, random_array_simulation も大きな配列を使用していることがわかる. 配列の名前とサイズは同じであるが, 別コア上の異なる LynxOS-178 インスタンスで動作しているため, この二つの配列は同一の変数ではない. 関数 data_creation_simulation は, あるコアで独立して動作し, random_array_simulation は別のコアで独立して動作するのであるが, 両者が並行動作し, 双方が L2 キャッシュを使う大容量データにアクセスすると干渉が生じる. これが表 I で見られる干渉であり, データサイズが増え, 多くのタイミング結合が起こると, 干渉も増加することになる.

## VI. AC 20-193 との関連性

表 III は AC 20-193 のオブジェクティブを満たすチェック項目とテストケースの一覧である. MCP_Resource_Usage_1, MCP_Resource_Usage_4, MCP_Software_2 はマッピングされたテストケースで満たされる. オブジェクティブ MCP_Software_1 について, ソフトウェア要件が青アプリにタイミング制約を課し, テストシナリオのいくつかで最悪実行時間がその要件を満たさなければ, ソフトウェアはこのオブジェクティブを満たせないことになる.

表 I.　赤アプリによる干渉がある場合のテスト対象関数の平均実行時間 (ns)

| 青アプリの配列サイズ | 赤アプリの配列サイズ | | | | | | |
|---|---|---|---|---|---|---|---|
| | *赤アプリなし* | *24 KB* | *256 KB* | *512 KB* | *768 KB* | *1 MB* | *2 MB* |
| 24 KB | 7627 ns | 7631 ns | | 7628 ns | | 7626 ns | 7640 ns |
| 256 KB | 93271 ns | 93278 ns | 96971 ns | 100417 ns | 104245 ns | 106311 ns | 102147 ns |
| 512 KB | 188737 ns | 192793 ns | 221364 ns | 247268 ns | 258706 ns | 263452 ns | 265848 ns |
| 768 KB | 340993 ns | 345350 ns | 386598 ns | 402369 ns | 408775 ns | 411370 ns | 413353 ns |
| 1 MB | 503886 ns | 508794 ns | 556454 ns | 577954 ns | | 588423 ns | 593055 ns |
| 2 MB | 1168882 ns | 1257074 ns | 1363913 ns | 1436625 ns | | 1476756 ns | 1392537 ns |

表 II.　タイミング結合 (関数のオーバーラップ)

| 関数 1 | 関数 2 | オーバーラップ開始時間(ns) | オーバーラップ終了時間(ns) | オーバーラップ時間(ns) |
|---|---|---|---|---|
| data_creation_simulation | random_array_simulation | 739,843,963,283 | 739,844,023,581 | 60,298 |
| random_array_simulation | data_creation_simulation | 739,844,023,634 | 739,844,295,438 | 271,804 |
| random_array_simulation | data_creation_simulation | 739,844,295,490 | 739,844,567,590 | 272,100 |
| random_array_simulation | data_creation_simulation | 739,844,567,640 | 739,844,837,329 | 269,689 |
| random_array_simulation | data_creation_simulation | 739,844,837,380 | 739,845,113,853 | 276,473 |
| random_array_simulation | data_creation_simulation | 739,845,113,905 | 739,845,386,403 | 272,498 |
| random_array_simulation | data_creation_simulation | 739,845,386,455 | 739,845,656,282 | 269,827 |
| random_array_simulation | data_creation_simulation | 739,845,656,332 | 739,845,928,094 | 271,762 |
| random_array_simulation | data_creation_simulation | 739,845,928,145 | 739,845,958,336 | 30,191 |
| data_creation_simulation | random_array_simulation | 739,845,958,371 | 739,846,200,430 | 242,059 |
| random_array_simulation | data_creation_simulation | 739,846,200,483 | 739,846,472,574 | 272,091 |
| random_array_simulation | data_creation_simulation | 739,846,472,626 | 739,846,741,948 | 269,322 |

表 IV は, AC 20-193 オブジェクティブと関連する DO-178C オブジェクティブとのマッピング, およびその根拠を示すものである. AC 20-193 はマルチコアプロセッサ上で動作するアプリケーション向け DO-178C 準拠の勧告通達であるため, これらのマッピングを行うことは理にかなっている.

### *A. 緩和策*

テスト結果によれば, セーフティクリティカルな環境において干渉を低減し決定的な挙動を向上させる緩和策の一つは, L2 キャッシュサイズに対するデータサイズを小さくすることである. データサイズが L1 キャッシュに収まる場合, タイミング変化は通常の実行での変動範囲内に収まるため無視できることが示されている.

しかし, データサイズが L1 キャッシュを超えると, L2 キャッシュを使うようになり干渉が増加する. 本論文では検討していないが, 干渉の緩和に関連する他の技法として, キャッシュ

分割やキャッシュカラーリング,L2キャッシュの無効化, RTOSによる堅牢な分割などが存在する.

表 III. AC 20-193オブジェクティブとマッピングされたテストケース

| オブジェクティブ | 項目名 | チェックリスト項目 | テストケース |
|---|---|---|---|
| MCP_Resource_Usage_1 | MCP構成設定 | - すべてのMCP構成はドキュメント化されているか？<br>- データ/制御結合に影響する設定はあるか(キャッシュ, コアアフィニティなど)？<br>- それらの設定は検証活動全体で一貫しているか？ | TCI_002 |
| MCP_Resource_Usage_4 | リソース割当て | - 共有MCPリソース(キャッシュ, インターコネクト, メモリ) はすべて特定されているか？<br>- ソフトウェアリソースへの要求は制限内に収まっていると検証されているか？<br>- リソース競合はコア間のデータ/制御結合に影響を与える可能性があるか？ | TCI_001, TCI_003 |
| MCP_Software_1 | 機能の実行 | - すべてのソフトウェアコンポーネントはMCPのフル負荷下で検証済みか？<br>- 負荷下での実行タイミングは正しい制御動作を維持しているか？<br>- 制御パスは予測可能で, 実行間で繰返し可能か？ | TCI_003 |
| MCP_Software_2 | 結合の検証 | - コア/コンポーネント間のすべてのデータ/ 制御結合が特定されているか？<br>- すべての結合は要件ベースのテスト中に実行されているか？<br>- 最終のシステム構成で結合動作が正しいことが検証されているか？ | TCI_004 |

表 IV. AC 20-193オブジェクティブ, DO-178Cへのマッピング, 根拠

| AC 20-193 オブジェクティブ | 関連するDO-178C オブジェクティブ | 根拠 |
|---|---|---|
| MCP_Resource_Usage_1 | [表A-7] A-7.4.a, A-7.4.d | 構成制御, 初期化データ, リソース使用制約 |
| MCP_Resource_Usage_4 | A-7.4.e (資源要件), A-7.10.b(分割) | 割当て検証, コア間に跨る競合が結合に影響を与える |
| MCP_Software_1 | A-7.2.c, A-7.4.d, A-7.10.d | 実行コンテキスト(タイミング制御, 機能的正確性)においてソフトウェアが正しく動作することを保証する |
| MCP_Software_2 | A-7.4.d (データ/制御結合), A-7.4.k(要件ベーステスト) | 最終構成の下で, 結合カバレッジと正しさを直接的に強制する |

## VII. 結論と今後の展開

本論文では, さまざまな適用シナリオを提示し, その結果から重大な干渉の影響を明らかにした. これは, システムの安全性を確保するためにAC 20-193で概説されたオブジェクティブを達成することの重要性を強調するものである. さらに, これらのオブジェクティブへの準拠を実証する方法を提案している. アプリケーションの観点では, 隠れたキャッシュ干渉が発生する可能性があり, その影響は比較的小さい場合(数パーセント)から, 約41%に達する比較的大きなものまで存在する. AC 20-193のオブジェクティブMCP_Resource_Usage_4は, 対象となるミッションの安全要件に適合するターゲットハードウェア環境を選択・設定するための指針として活用できる. さらに, MCP_Software_2に準拠し, 複数のコア上で動作する複数の独立したアプリケーション間におけるデータ結合, 制御結合, タイミング結合を解析することで, コードの実行と制御フローの観点で, オブジェクティブMCP_Software_1を満たすためにテストすべき最悪シナリオを特定できる. それによって, 最悪の実行シナリオがセーフティクリティカルなタスクに要求されるタイミングを超えないことを検証できる.

本論文ではメモリ, 特にL2キャッシュ干渉とデータサイズの緩和に焦点を当てた. 今後の研究では, キャッシュ分割, キャッシュカラーリング, L2キャッシュ無効化, RTOSによる堅牢な分割など, L2 キャッシュ干渉の緩和の他の手法を検討する. また, 潜在的な干渉点として, 異なるハードウェア共有リソースについても調査する. さらに, 人工知能(AI)や機械学習(ML)アルゴリズムの普及が進む中, データ結合・制御結合やタイミング結合をどのように適用し, そのアルゴリズムが機能安全規格に準拠することを保証する方法について探求する予定である.

## 謝辞

アメリカ陸軍戦闘能力開発司令部航空・ミサイルセンターは, 航空・ミサイル能力, およびライフサイクルエンジニアリングソリューションを通じて国家の戦闘要員に対する対応力を高めている.

## 参考文献

[1] Federal Aviation Administration (FAA), Advisory Circular AC 20-193: Use of Multi-Core Processors, U.S. Department of Transportation, Jan. 8, 2024.

[2] RTCA DO-178C. Software Considerations in Airborne Systems and Equipment Certification, RTCA, Inc., 2011.

[3] Pressman, R. S. (2014). *Software Engineering: A Practitioner's Approach*. McGraw-Hill Education.

[4] IEEE Std 610.12-1990. IEEE Standard Glossary of Software Engineering Terminology, IEEE, 1990.

[5] ISO/IEC/IEEE 24765:2017. *Systems and software engineering — Vocabulary*, ISO, 2017.

[6] LDRA. “LDRA Tool Suite.” [Online]. Available: https://ldra.com/products/ldra-tool-suite/.

[7] RTCA, Inc., DO-330: Software Tool Qualification Considerations, Washington, D.C., DFec. 2011.

[8] EASA, “AMC 20-193 Use of multi-core processors,” 2022. [Online]. Available: https://www.easa.europa.eu/sites/default/files/dfu/annex_i_to_ed_decision_2022-001-r_amc_20-193_use_of_multi-core_processors_mcps.pdf.

[9] EUROCAE , “ED-12C - Software considerations in airborne systems and equipment certification – with Corrigendum 1”, refer to: https://www.eurocae.net/product/ed-12c-corr-1-software-considerations-in-airborne-systems-and-equipment-certification-corrigendum-1/.

[10] U.S. Army DEVCOM. “Army Military Airworthiness Certification Criteria (AMACC), Revision B” November 13, 2024.

[11] Lynx Software Technologies. “LynxOS-178: D0-178C & DO-356 Certified RTOS for Avionics & Safety-Critical Systems.” [Online]. Available: https://www.lynx.com/products/lynxos-178-do-178c-certified-posix-rtos.

[12] W. Vance, R. Ashok, J. Ross, B. Jacobs, T. Bui and M. Wotell, "Analysis of Cache Memory Interference on Multicore Systems Utilizing Shared Memory Aggressor Applications," 2024 AIAA DATC/IEEE 43rd Digital Avionics Systems Conference (DASC), San Diego, CA, USA, 2024, pp. 1-6, doi: 10.1109/DASC62030.2024.10749008.

[13] Vance, William & Mott, Michael & Wotell, Mark & Bui, Tuan & Ross, John. (2023). "Insights from Preliminary Analysis of Local Cache Performance in COTS RTOS for Multi-Core Processors." 1-8. 10.1109/DASC58513.2023.10311340.

[14] R. Mancuso, R. Dudko, E. Betti, M. Cesati, M. Caccamo and R. Pellizzoni, "Real-time cache management framework for multi-core architectures," 2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS), Philadelphia, PA, USA, 2013, pp.45-54, doi: 10.1109/RTAS.2013.6531078.

[15] Lynx Software Technologies. (n.d.). "LynxSecure Separation Kernel Hypervisor." [Online]. Available: https://www.lynx.com/products/lynxsecure-separation-kernel-hypervisor

[16] LDRA. "Multicore processors and critical embedded systems: WCET, interference research, and other challenges." [Online]. Available: https://ldra.com/capabilities/mcp/.

[17] International Organization for Standardization, ISO 26262:2018 "Road vehicles – Functional safety", International Organization for Standardization, 2018.

[18] International Electrotechnical Commission (IEC), IEC 61508 Functional safety of electrical/electronic/programmable electronic safety-related systems, IEC, 2010 (Part 1).

[19] M. Woodside, G. Franks and D. C. Petriu, "The Future of Software Performance Engineering," Future of Software Engineering (FOSE '07), Minneapolis, MN, USA, 2007, pp. 171-187, doi: 10.1109/FOSE.2007.32.

[20] Ross, John & Vance, William & Ashok, Roshini & Jacobs, Bruce & Bui, Tuan & Wotell, Mark. (2024). "Analyzing Shared Cache Partitioning on an NXP P4080 Processor as a Method of Multi-Core Interference Mitigation." 1-5. 10.1109/DASC62030.2024.10748905.