

## 27. 組み込みシステム最適化に関する知識 I

### 1. 科目の概要

システムの性能を最適化するための方法としてのマルチプロセッサ導入や、リアルタイムシステムの特徴を活かした最適化、ソフトウェア最適化や最適化のためのシステム評価方法など、組み込みシステムの最適化に関する様々な技術について解説する。

### 2. 習得ポイント

本科目の学習により習得することが期待されるポイントは以下の通り。

習得ポイント	説明	シラバスの対応コマ
I-27-1. マルチプロセッサシステムの種類	マルチプロセッサシステムについて、その基本概念と特徴、利点などを示し、従来からの分類方法であるSISD/SIMD/MISD/MIMDといった種別や、新しい分類方法であるSMP (Symmetric Multi Processor)、クラスタ構成、MPP (Massively Parallel Processor)などを紹介する。	1
I-27-2. マルチプロセッサシステム活用上の留意点、構成例	タスクのスケジューリング、プロセッサ間の同期と通信、ハードウェアの制限など、マルチプロセッサシステムを活用する際に留意すべき事項を説明する。またマルチプロセッサシステムの具体的な構成例を紹介し、その特徴について説明する。	1
I-27-3. マルチプロセッサによるシステムの最適化	性能の最適化方法、CPU高速化、入出力方法に関する工夫、低電力処理など、マルチプロセッサを使用したシステムを最適化する手順や配慮すべき項目について解説する。	2
I-27-4. リアルタイムシステムの設計と留意点	リアルタイムシステムの設計に必要な要素を解説する。具体的には、モジュール分割、処理分割による最適化、割り込み対応、処理時間・応答時間の見積り、処理待ちキュー設計、デバイスに対する高速応答処理の実装などについて触れる。	3
I-27-5. タスク分割による性能向上	リアルタイムソフトウェアを利用したシステムの性能向上、最適化のためのソフトウェア作成方法を解説する。特に適切なタスク分割によるタスクの同期、並行動作の実現など、関連する話題を説明する。	4
I-27-6. タスク関連設計によるリアルタイムシステムの最適化	タスク関連設計を中心とした最適化のアプローチについて解説する。タスク関連設計の基本的な考え方と特徴について説明し、さらに、タスク関連図の作成方法、タスク優先順位の決定、起動タイミング、タスク間の同期、排他制御といった項目について説明する。	4
I-27-7. リアルタイム処理に対する評価項目と留意点	リアルタイムシステムの性能評価について、全体のパフォーマンス、処理のスループット、全体の優先順位、デバイスドライバの処理方法などの観点からみた評価項目と評価時の留意点を説明する。またタスク分割の設計評価についても解説する。	5
I-27-8. CPUの性能指標と評価方法	CPI (Clock Cycles Per Instruction)、MIPS (Million Instructions Per Second)など、CPUの性能を評価する指標を紹介する。ただし様々なシステムの利用状況においては、これらの指標だけでは不十分であり個別の最適化手法を適用しなければならないことについても言及する。	6
I-27-9. ソースコード最適化によるシステム性能向上	ソフトウェア最適化によるシステムの性能向上について、基本的な概念と設計方針を解説する。ソフトウェア最適化の基本手段である冗長なコードの排除やループの展開、インライン化、変数のレジスタ割り当てなど個別の手法を紹介し、またソースコードレベルの最適化の手順についても説明する。	6
I-27-10. プログラムモジュール配置による最適化	プログラムモジュールの配置方法に係る最適化手法を解説する。プログラムモジュールのメモリへのローディング方法、リンク方式、常駐化など具体的な方法について説明する。	6

#### 【学習ガイダンスの使い方】

- 「習得ポイント」により、当該科目で習得することが期待される概念・知識の全体像を把握する。
- 「シラバス」、「IT 知識体系との対応関係」、「OSS モデルカリキュラム固有知識」をもとに、必要に応じて、従来の IT 教育プログラム等との相違を把握した上で、具体的な講義計画を考案する。
- 習得ポイント毎の「学習の要点」と「解説」を参考にして、講義で使用する教材等を準備する。

### 3. IT 知識体系との対応関係

「27. 組み込みシステム最適化に関する知識 I」と IT 知識体系との対応関係は以下の通り。

科目名	基本レベル(1)														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
27. 組み込みシステム最適化に関する知識	<マルチプロセッサシステム>	<ハードウェアによる最適化>	<リアルタイムシステムの設計>	<リアルタイムソフトウェアの条件と最適化>	<性能最適化の評価項目>	<ソフトウェアの最適化>	<MPUの性能最適化設計>	<システムの性能要件と評価項目>	<システム性能の評価方法>	<性能評価手法の分類>	<拡張性の評価>	<システム資源のトレードオフ>	<基本ソフトウェアと応用ソフトウェアのトレードオフ>	<組み込みシステム最適化のための方式設計>	<最適化のための検討項目>

[シラバス : [http://www.ipa.go.jp/software/open/oss/download/Model\\_Curriculum\\_05\\_27.pdf](http://www.ipa.go.jp/software/open/oss/download/Model_Curriculum_05_27.pdf)]

#### <IT 知識体系上の関連部分>

分野	科目名	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
組込みシステム開発の基礎	1	IT-IAS. 情報保証と情報セキュリティ	IT-IAS1. 基礎的知識	IT-IAS2. 情報セキュリティの仕組み(対策)	IT-IAS3. 運用上の問題	IT-ISA. ホリゾ	IT-IAS5. 攻撃	IT-IAS6. 情報セキュリティ分野	IT-IAS7. フォレンジック(情報保証)	IT-IAS8. 情報の保証	IT-IAS9. 情報セキュリティサービス	IT-IAS10. 脅威分析モデル	IT-IAS11. 脆弱性			
	2	IT-SP. 社会的な観点とプロフェッショナルとしての課題	IT-SP1. プロフェッショナルとしてのコミュニケーション	IT-SP2. コンピュータの歴史	IT-SP3. コンピュータを取り巻く社会環境	IT-SP4. チームワーク	IT-SP5. 知的財産権	IT-SP6. コンピュータの法的問題	IT-SP7. 組織の中のIT	IT-SP8. プロフェッショナルとしての倫理的な問題と責任	IT-SP9. プライバシーと個人の自由					
応用技術	3	IT-IM. 情報管理	IT-IM2. 情報管理の概念と基礎	IT-IM2. データベース問合わせ言語	IT-IM3. データアーキテクチャ	IT-IM4. データモデリングとデータベース設計	IT-IM5. データと情報の管理	IT-IM6. データベースの応用分野								
	4	IT-WS. Webシステムとその技術	IT-WS1. Web技術	IT-WS2. 情報アーキテクチャ	IT-WS3. デジタルメディア	IT-WS4. Web開発	IT-WS5. 脆弱性	IT-WS6. ソーシャルソフトウェア								
ソフトウェアの方法と技術	5	IT-PF. プログラミング基礎	IT-PF1. 基本データ型	IT-PF2. プログラミングの基本的構成要素	IT-PF3. オブジェクト指向プログラミング	IT-PF4. アルゴリズムと問題解決	IT-PF5. イベント駆動プログラミング	IT-PF6. 再帰								
	6	IT-PT. 技術を統合するためのプログラミング	IT-PT1. システム連携	IT-PT2. データ取り扱って交換	IT-PT3. 統合的プログラミング	IT-PT4. スクリプトプログラミング	IT-PT5. ソフトウェアセキュリティの実現	IT-PT6. 種々の保護	IT-PT7. プログラミング言語の概要							
	7	DE-SME. ソフトウェア工学	DE-SME0. 歴史と概要	DE-SME1. ソフトウェアプロセス	DE-SME2. ソフトウェアの要求と仕様	DE-SME3. ソフトウェアの設計	DE-SME4. ソフトウェアのテストと検証	DE-SME5. ソフトウェアの保守	DE-SME6. ソフトウェアプロジェクト管理	DE-SME7. ソフトウェアプロジェクト管理	DE-SME8. 言語翻訳	DE-SME9. ソフトウェアのフォーマットトランスラタ	DE-SME10. ソフトウェアの構成管理	DE-SME11. ソフトウェアの標準化		
	8	IT-SIA. システムインテグレーションとアーキテクチャ	IT-SIA1. 要求仕様	IT-SIA2. 調達/手配	IT-SIA3. インテグレーション	IT-SIA4. プロジェクト管理	IT-SIA5. テストと品質保証	IT-SIA6. 組織の特性	IT-SIA7. アーキテクチャ							
システム基盤	9	IT-NET. ネットワーク	IT-NET1. ネットワークの基礎	IT-NET2. ルーティングとスイッチング	IT-NET3. 物理層	IT-NET4. セキュリティ	IT-NET5. アプリケーション分野	IT-NET6. ネットワーク管理								
	10	DE-NWK. テレコミュニケーション	DE-NWK0. 歴史と概要	DE-NWK1. 通信ネットワークのアーキテクチャ	DE-NWK2. 通信ネットワークのプロトコル	DE-NWK3. LANとWAN	DE-NWK4. クラウドサーバコンピュティン	DE-NWK5. データのセキュリティと整合性	DE-NWK6. ワイヤレスコンピューティングとモバイルコンピューティング	DE-NWK7. データ連携	DE-NWK8. 組み込み機器向けネットワーク	DE-NWK9. 通信技術とネットワーク	DE-NWK10. 性能評価	DE-NWK11. ネットワーク管理	DE-NWK12. 圧縮と伸張	
	11	IT-PI. オペレーティングシステム	IT-PI1. オペレーティングシステム	IT-PI2. アーキテクチャと機構	IT-PI3. コンピュータインフラストラクチャ	IT-PI4. デプロイメントソフトウェア	IT-PI5. ファームウェア	IT-PI6. ハードウェア								
アプリケーション	12	DE-OPS. オペレーティングシステム	DE-OPS0. 歴史と概要	DE-OPS1. 並行性	DE-OPS2. スケジューリングと管理	DE-OPS3. メモリ管理	DE-OPS4. セキュリティと保護	DE-OPS5. ファイル管理	DE-OPS6. リアルタイムOS	DE-OPS7. OSの概要	DE-OPS8. 設計の原則	DE-OPS9. デバイス管理	DE-OPS10. システム性能評価			
	13	DE-CA0. コンピュータアーキテクチャと構成	DE-CA00. 歴史と概要	DE-CA01. コンピュータアーキテクチャの基礎	DE-CA02. メモリシステムの構成とアーキテクチャ	DE-CA03. インタフェースと通信	DE-CA04. デバイスサブシステム	DE-CA05. CPUアーキテクチャ	DE-CA06. 性能・コスト評価	DE-CA07. 分岐・並列処理	DE-CA08. コンピュータによる計算	DE-CA09. 性能向上				
複数領域にまたがるもの	14	IT-ITF. IT基礎	IT-ITF1. IT00-組織的なテーマ	IT-ITF2. 組織の問題	IT-ITF3. IT1の歴史	IT-ITF4. IT分野(学科)とそれに関連する分野(学科)	IT-ITF5. 応用情報	IT-ITF6. IT分野における数学と統計学の活用								
	15	DE-ESY. 組み込みシステム	DE-ESY0. 歴史と概要	DE-ESY1. 低電力コンピューティング	DE-ESY2. 高信頼性システムの設計	DE-ESY3. 組み込みアーキテクチャ	DE-ESY4. 開発環境	DE-ESY5. ライフサイクル	DE-ESY6. 要件分析	DE-ESY7. 仕様定義	DE-ESY8. 構造設計	DE-ESY9. テスト	DE-ESY10. プロジェクト管理	DE-ESY11. 並行設計(ハードウェア、ソフトウェア)	DE-ESY12. 実装	

### 4. OSS モデルカリキュラム固有の知識

OSS モデルカリキュラム固有の知識として、組み込みシステム開発におけるタスク設計手法と最適化のための評価方法がある。タスク設計の評価や MPU の評価など実践的な内容を扱う。

科目名	第1回	第2回	第3回	第4回	第5回	第6回
27. 組み込みシステム最適化に関する知識 I	(1)マルチプロセッサの分類 (2)マルチプロセッサシステムの活用方法 (3)マルチプロセッサシステムの活用方法 (4)マルチプロセッサシステムの構成例	(1)性能の最適化 (2)CPU 高速化の仕組み (3)入出力方式 (4)低電力処理	(1)リアルタイムシステム設計 (2)アプリケーションタスクの設計 (3)デバイスプログラミング	(1)カーネル処理とドライバ (2)タスク設計 (3)入出力単位 (4)タスク関連設計による最適化のアプローチ	(1)評価項目と留意点 (2)タスク分割の設計評価 (3)MPU の評価指標ごとの評価方法	(3)MPU の評価指標ごとの評価方法 (2)プログラムと処理 (3)最適化コンパイラ

(網掛け部分は IT 知識体系で学習できる知識を示し、それ以外は OSS モデルカリキュラム固有の知識を示している)

スキル区分	OSS モデルカリキュラムの科目	レベル
組み込み分野	27 組み込みシステム最適化に関する知識 I	基本
習得ポイント	I-27-1. マルチプロセッサシステムの種類	
対応する コースウェア	第 1 回 (マルチプロセッサシステム)	

## I-27-1. マルチプロセッサシステムの種類

マルチプロセッサシステムについて、その基本概念と特徴、利点などを示し、従来からの分類方法である SISD/SIMD/MISD/MIMD といった種別や、新しい分類方法である SMP (Symmetric Multi Processor)、クラスタ構成、MPP (Massively Parallel Processor)などを紹介する。

### 【学習の要点】

- \* マルチプロセッサシステムでは、処理を一つの CPU ではなく、複数の CPU で並列して行う。これにより処理力の向上と耐故障性の向上とを実現する。
- \* マルチプロセッサの分類として、プロセッサの数とデータを並行処理する方式による SISD/SIMD/MISD/MIMD の分類が従来から使われている。
- \* 対称的(Symmetric Multi Processor: SMP)、非対称的(Asymmetric Multi Processor: AMP)といった分類の他、超並列(Massively Parallel Processor:MPP)などの分類が使われる。

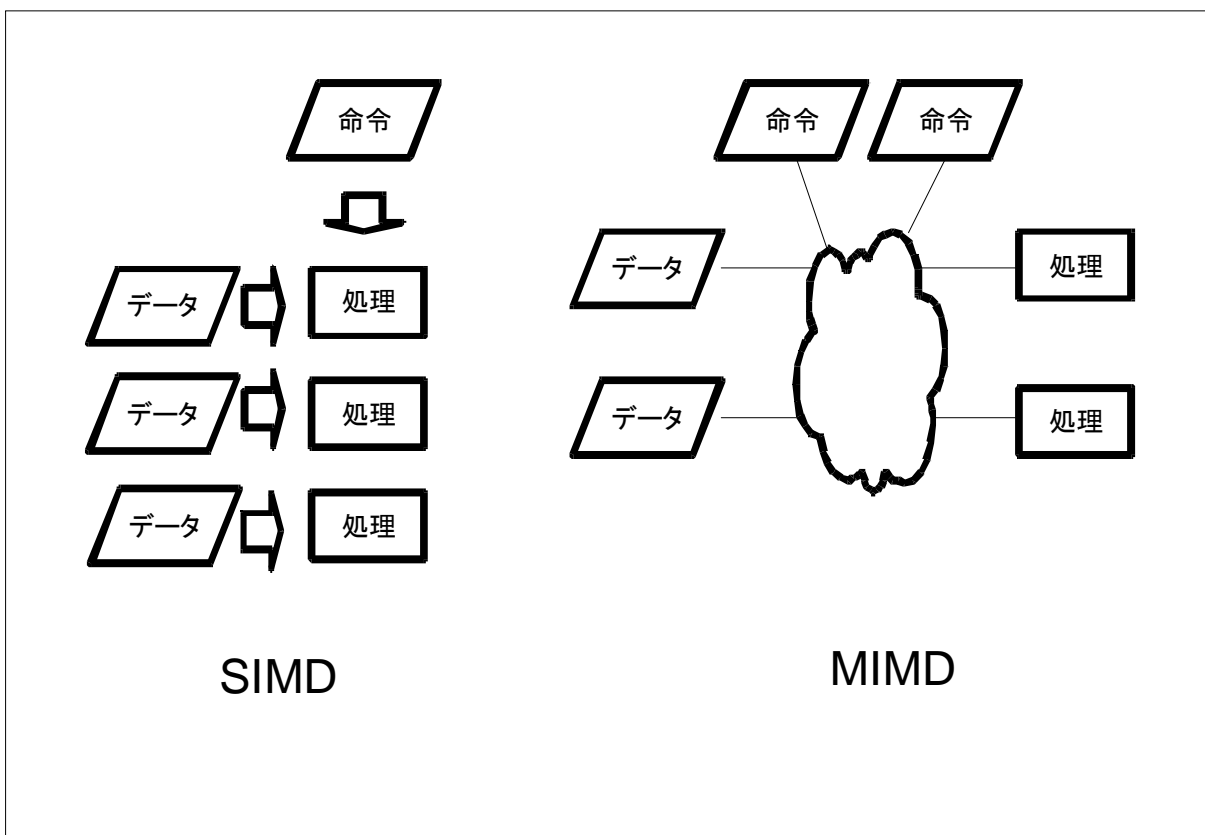


図 I-27-1. SIMD と MIMD

## 【解説】

### 1) マルチプロセッサシステム

#### \* マルチプロセッサの分類 (Flynn の分類)

- SISD (Single Instruction stream Single Data stream: 単一命令単一データフロー)  
従来型のコンピュータの処理形式であり、1 命令で 1 つのデータフローを処理する。
- SIMD (Single Instruction stream Multiple Data stream: 単一命令複数データフロー)  
1 命令で 1 つの配列を処理する。代表例としてベクトルプロセッサがある。
- MISD (Multiple Instruction stream Single Data stream: 複数命令単一データフロー)  
複数のプロセッサで同じ 1 つのデータフローの処理を行う。冗長性があり、高度な信頼性を求められる場合にのみ使われる。
- MIMD (Multiple Instruction stream Multiple Data stream: 複数命令複数データフロー)  
複数の独立したプロセッサが、同時に異なるプログラムを処理する。

#### \* 対称型・非対称型

- 対称マルチプロセッサ(SMP)は MIMD アーキテクチャの例である。一般に SMP では、同一種類のプロセッサが用いられ、どのプロセッサも他のプロセッサと同じ計算ができる。
- 非対称マルチプロセッサ(AMP)では、複数種類のプロセッサが用いられる。各プロセッサが処理用途に応じた能力を持っている場合が多い。

#### \* 超並列コンピュータ

- MPP では、複数のマイクロプロセッサを用いて並列処理を行う。高性能な専用プロセッサを使うスーパーコンピュータよりも安価に高速化できる。

スキル区分	OSS モデルカリキュラムの科目	レベル
組み込み分野	27 組み込みシステム最適化に関する知識 I	基本
習得ポイント	I-27-2. マルチプロセッサシステム活用上の留意点、構成例	
対応する コースウェア	第1回 (マルチプロセッサシステム)	

## I-27-2. マルチプロセッサシステム活用上の留意点、構成例

タスクのスケジューリング、プロセッサ間の同期と通信、ハードウェアの制限など、マルチプロセッサシステムを活用する際に留意すべき事項を説明する。またマルチプロセッサシステムの具体的な構成例を紹介し、その特徴について説明する。

### 【学習の要点】

- \* タスクのスケジューリング、プロセッサ間の同期と通信、ハードウェアの制限など、マルチプロセッサシステムを活用する際に留意すべき事項を説明する。またマルチプロセッサシステムの具体的な構成例を紹介し、その特徴について説明する。
- \* マルチプロセッサシステムを用いることにより、複数事象の制御、タスクのスケジューリング、プロセッサ間の同期・通信について考慮する必要が生じる。
- \* マルチプロセッサシステムは、汎用 CPU2 個による小さな構成のものから、数万個の CPU を接続した超並列コンピュータまで幅広く存在する。

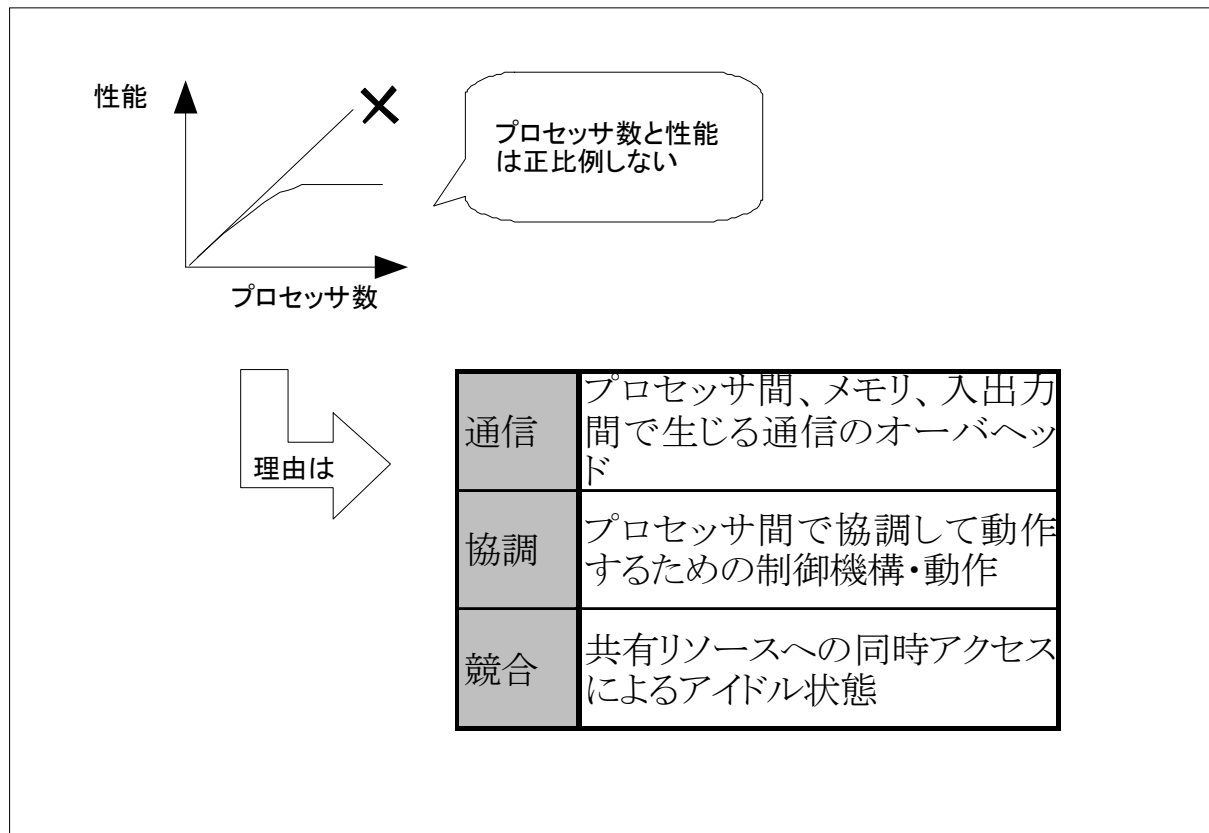


図 I-27-2. マルチプロセッサの性能低下要因

## 【解説】

### 1) マルチプロセッサシステムの活用方法

- \* ロックと相互排除
  - マルチプロセッサの環境では、ハードウェアロックが用いられる。
  - プログラマは相互排除のため、明示的にロックを利用する必要がある。
- \* 対称・非対称マルチプロセッサでのプログラミング
  - 対称プロセッサのプログラミングの方が非対称プロセッサの場合よりも容易とされる。
  - プログラマが理解する必要のあるプロセッサの命令セットが 1 種類で済む、タスクをどのプロセッサに割り当てるか考慮せずに済む、などの理由による。

### 2) マルチプロセッサシステムの性能への障壁

直観的にはマルチプロセッサシステムの処理能力の方が、単一プロセッサアーキテクチャの能力よりも優る。しかし、N 個の CPU により処理をすることにより速度が N 倍になるわけではない。性能の障壁となる要因は以下の 3 点が挙げられる。

- \* 通信
  - プロセッサ間、メモリ、入出力間で生じる通信のオーバーヘッドは無視できないものとなる。
- \* 協調
  - プロセッサ間で協調して動作するための制御機構が必要になる。
- \* 競合
  - 共有リソースへの同時アクセスによりタスクがアイドル状態にされる。これにより劇的な性能低下を招く。

スキル区分	OSS モデルカリキュラムの科目	レベル
組み込み分野	27 組み込みシステム最適化に関する知識 1	基本
習得ポイント	I-27-3. マルチプロセッサによるシステムの最適化	
対応する コースウェア	第 2 回 (ハードウェアによる最適化)	

### I-27-3. マルチプロセッサによるシステムの最適化

性能の最適化方法、CPU 高速化、入出力方法に関する工夫、低電力処理など、マルチプロセッサを使用したシステムを最適化する手順や配慮すべき項目について解説する。

#### 【学習の要点】

- \* CPU 性能の高速化のためにパイプライン処理が行われる。パイプライン処理により複数の命令が並列して処理されていく。
- \* ハードウェアによるパイプラインには、命令パイプラインとデータパイプラインがある。
- \* 並列処理、多段パイプライン、低電圧動作の組み合わせによる低電力処理が行われてきた。

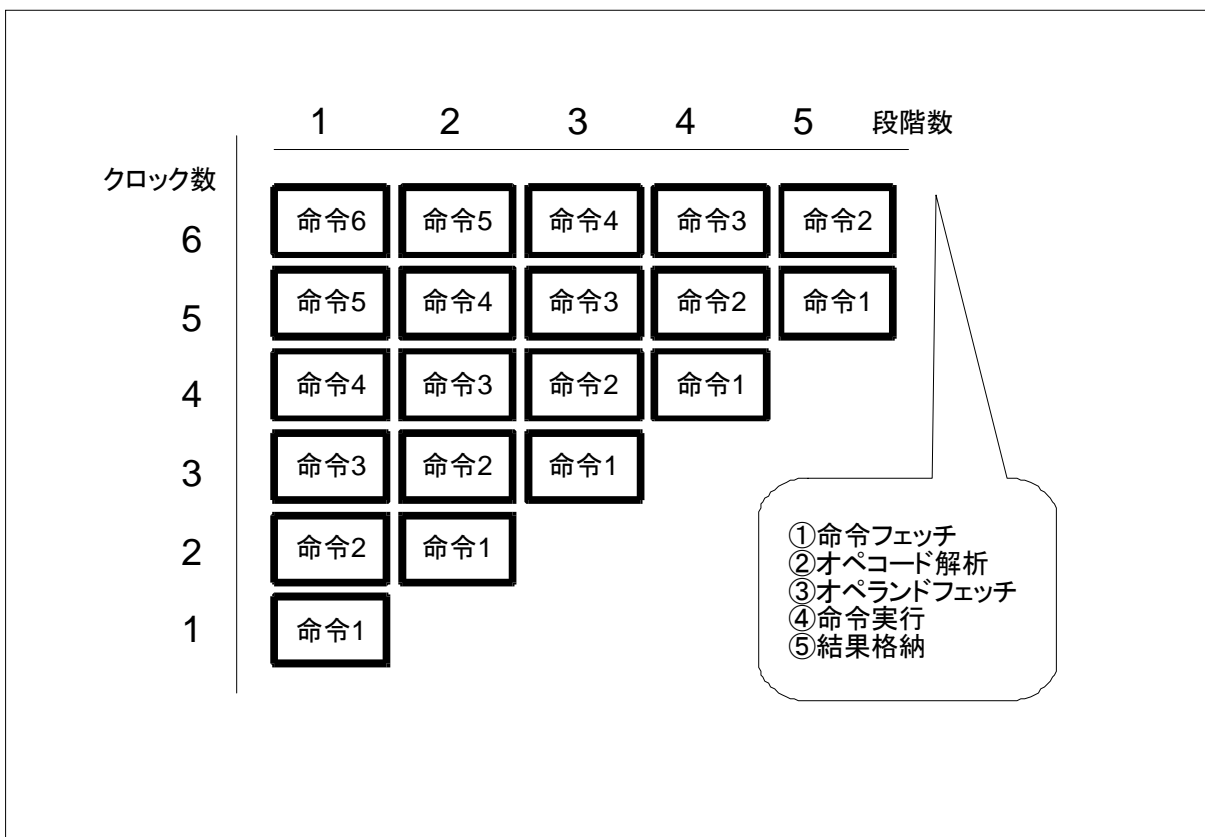


図 I-27-3. パイプライン処理(命令パイプライン)

## 【解説】

### 1) パイプライン処理による CPU 高速化の仕組み

RISC プロセッサでパイプライン処理を利用することにより、1 クロックサイクルで命令フェッチ-実行サイクルの全てのステップを実行するのと同等の処理ができる。

- \* プロセッサが命令をフェッチし実行するサイクルに対してパイプライン処理を行う場合、一般に以下のモデルで説明される。
  - 命令フェッチ
  - オペコード解析
  - オペランドフェッチ
  - 命令実行
  - 結果格納
- \* 命令を流れ作業のようにパイプラインを通して実行させることにより、常に 5 つの命令がパイプラインに含まれることとなる。

### 2) 低電力処理

- \* 動作電力は、デバイスの動作を通じて消費される電力のことで、周波数、電圧、負荷に依存する。
- \* 多段パイプラインによる並列処理でスループットを上げることにより、結果として消費電力を抑制できる。
- \* マルチプロセッサは動作領域を制限できるため平均消費電力を下げる可以降低。

スキル区分	OSS モデルカリキュラムの科目	レベル
組み込み分野	27 組み込みシステム最適化に関する知識 I	基本
習得ポイント	I-27-4. リアルタイムシステムの設計と留意点	
対応する コースウェア	第 3 回 (リアルタイムシステムの設計)	

## I-27-4. リアルタイムシステムの設計と留意点

リアルタイムシステムの設計に必要な要素を解説する。具体的には、モジュール分割、処理分割による最適化、割り込み対応、処理時間・応答時間の見積り、処理待ちキュー設計、デバイスに対する高速応答処理の実装などについて触れる。

### 【学習の要点】

- \* リアルタイムシステムの設計手順としては、全体のシステムを複数のモジュールに分割した上で、個々のモジュールに対しタスクの分割をし、その上で共有資源(ファイル、セマフォ、など)の設計を行う。
- \* タスク分割の指針としては、システム全体で時間的に最も緊急度の高い処理(タイムクリティカル処理)に注目する。タイムクリティカルな処理を中心にタスク分割を進めていく。

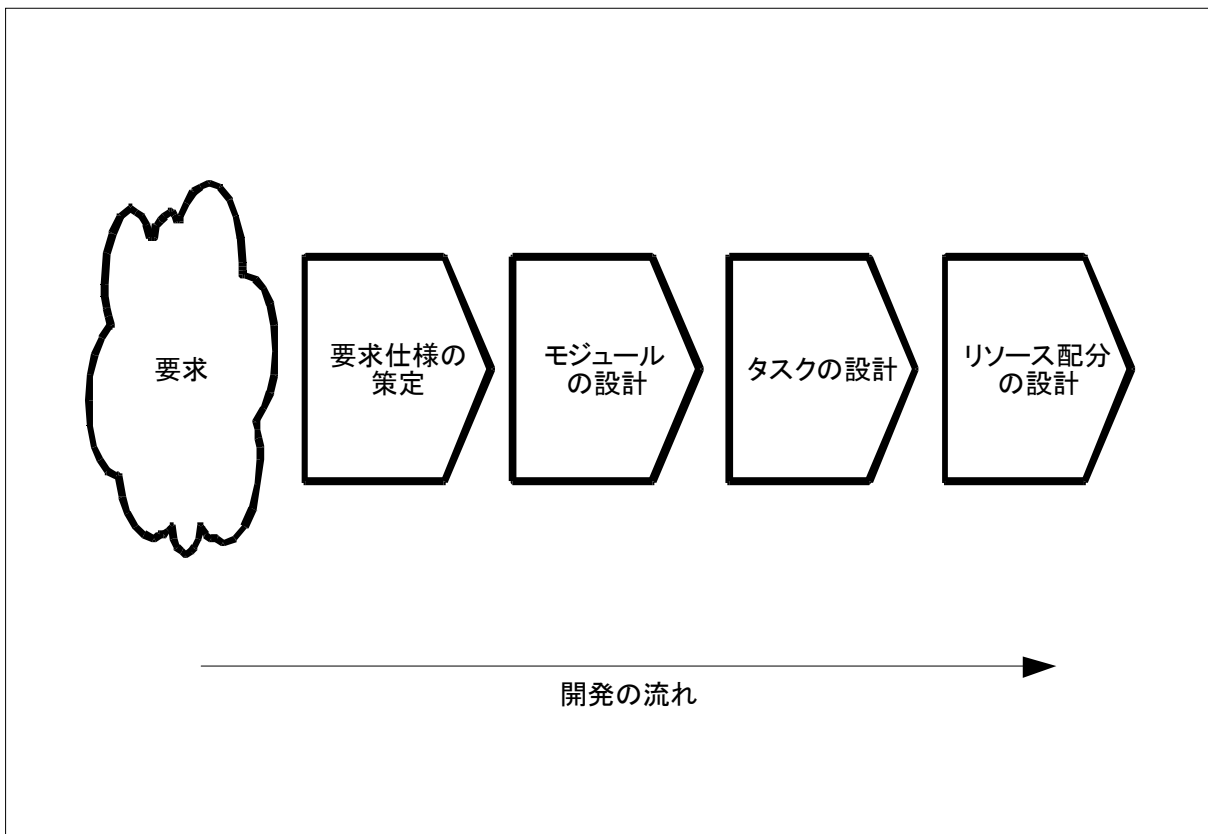


図 I-27-4. 設計手順の概略

## 【解説】

### 1) リアルタイムシステム設計

- \* 一般的なリアルタイムシステムの設計では、システムを実現する機能をタスクまで分割していく。
  - はじめに全体のシステムを複数のモジュールに分割する。それぞれのモジュールは機能に対応して分割される。
  - 個々のモジュールの中身をタスクへと分割する。
  - 共有リソース配分(ファイル、セマフォ、など)の設計を行う。
- \* タスク設計に関して絶対的な理論は存在しない。
- \* リアルタイム性を考慮する場合、割り込みハンドラなどを利用してタイムクリティカルな処理を中心にタスクの優先度を設定していく。
- \* タスクを起動するイベントは入出力のイベント、周期的なクロックイベント、他のタスクの処理によるイベントがある。こうした起動の要因ごとにタスクを分割することも指針として利用される。

### 2) デバイスプログラミング

- \* 割り込みハンドラの処理は、カーネルの設計によって異なる。割り込みが無い場合のみ高い性能を出せるカーネルや、割り込みがあっても極端に性能が落ちないカーネルなど様々である。
- \* また、カーネルインタフェースの設計によっては、割り込みハンドラからカーネルに対するシステムコールに制限をかける場合もある。
- \* デバイスドライバは入出力機能を実行する機能をタスクから独立する形で実装したソフトウェアである。
  - 組み込み開発においてデバイスドライバを作成する機会はエンタプライズ系に比べて一般に多くなる。
  - デバイスドライバはアプリケーションインターフェイスと割り込みハンドラのコンテキストから構成される。

スキル区分	OSS モデルカリキュラムの科目	レベル
組み込み分野	27 組み込みシステム最適化に関する知識 I	基本
習得ポイント	I-27-5. タスク分割による性能向上	
対応する コースウェア	第 4 回 (リアルタイムソフトウェアの条件と最適化)	

## I-27-5. タスク分割による性能向上

リアルタイムソフトウェアを利用したシステムの性能向上、最適化のためのソフトウェア作成方法を解説する。特に適切なタスク分割によるタスクの同期、並行動作の実現など、関連する話題を説明する。

### 【学習の要点】

- \* タスク設計の際には、入出力の速度差を解消すること、起動方式によってタスク分割を行うこと、割り込みハンドラとの処理分担を適切に行うことなども指針とされる。
- \* 入出力ポートの利用は、複数タスクから同時にはできない。そのため、あるタスクがポートを利用している間は、他のタスクからの入出力要求はキューで待機する。
- \* 並行処理を特定するためのガイドラインでは、「デバイス依存性の特定、イベント依存性の特定、時間依存性の特定(緊急度の度合い)、計算に特化したアクティビティの特定」(出典:Qing Li「リアルタイム組込み OS 基礎講座」、翔泳社(2005))などの手順に従う。

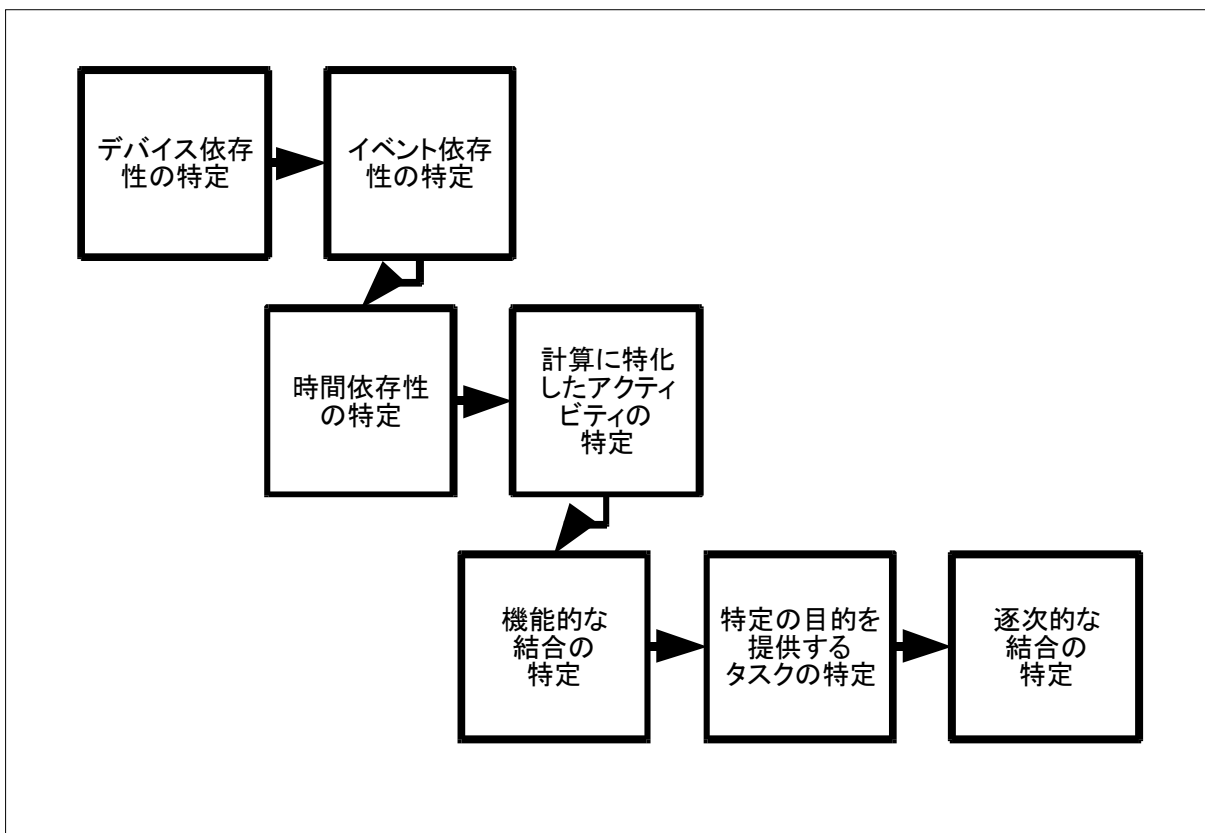


図 I-27-5. 並列処理を行うべき箇所を特定するガイドライン(出典:Qing Li「リアルタイム組込み OS 基礎講座」、翔泳社(2005)に基づき三菱総合研究所が作成)

## 【解説】

### 1) タスク設計

タスクを適切に分割し、優先度を割り当てることでリアルタイム性を確保することがタスク設計の一つの重要な方針である。

#### \* アウトサイドインアプローチ

- アプリケーションの分解を行う手法であり、システムの入出力を特定することで、デバイスへのインターフェイスや入出力の関係を矢印で結んだコンテキスト図を記述する。
- コンテキスト図の入出力から潜在的なタスクを図に追加していく。この情報をもとにアプリケーションを分割していく。
- この手法のように、タスク設計の指針として入出力の単位ごとにタスク分割を行うことがしばしば行われる。入出力の速度差をキューによって解消でき効率の良いシステムを作ることができる。

#### \* 並行処理を特定するためのガイドライン

アプリケーション内の処理の並行性を特定するために、定性的なタスク解析手順が存在する。

- デバイス依存性の特定  
入出力割り込みの有無、同期非同期かによりタスクの独立性が決定される。
- イベント依存性の特定  
イベントを適切に処理するタスク群が必要である。
- 時間依存性の特定(緊急度の度合い)  
緊急度の高い順に処理を割り当てるのが基本原則となる。
- 計算に特化したアクティビティの特定  
入出力処理など CPU 時間を多く取る処理が CPU を独占しないように優先度を下げる。
- 機能的な結合の特定  
機能的に関係の高い処理を一つのタスクとしてまとめることにより同期と通信のオーバーヘッドを避けることができる。
- 特定の目的を提供するタスクの特定
- 逐次的な結合の特定

スキル区分	OSS モデルカリキュラムの科目	レベル
組み込み分野	27 組み込みシステム最適化に関する知識 I	基本
習得ポイント	I-27-6. タスク関連設計によるリアルタイムシステムの最適化	
対応する コースウェア	第 4 回(リアルタイムソフトウェアの条件と最適化)	

## I-27-6. タスク関連設計によるリアルタイムシステムの最適化

タスク関連設計を中心とした最適化のアプローチについて解説する。タスク関連設計の基本的な考え方と特徴について説明し、さらに、タスク関連図の作成方法、タスク優先順位の決定、起動タイミング、タスク間の同期、排他制御といった項目について説明する。

### 【学習の要点】

- \* リアルタイムシステムの最適化のために、タスク関連図を用いた最適化が行われる。
- \* タスク関連図作成時の留意点としては、タスクの起動関係とタイミング、タスク間の同期とその手段、タスク間のキュー、重要なフラグの更新と参照の関連が挙げられる。

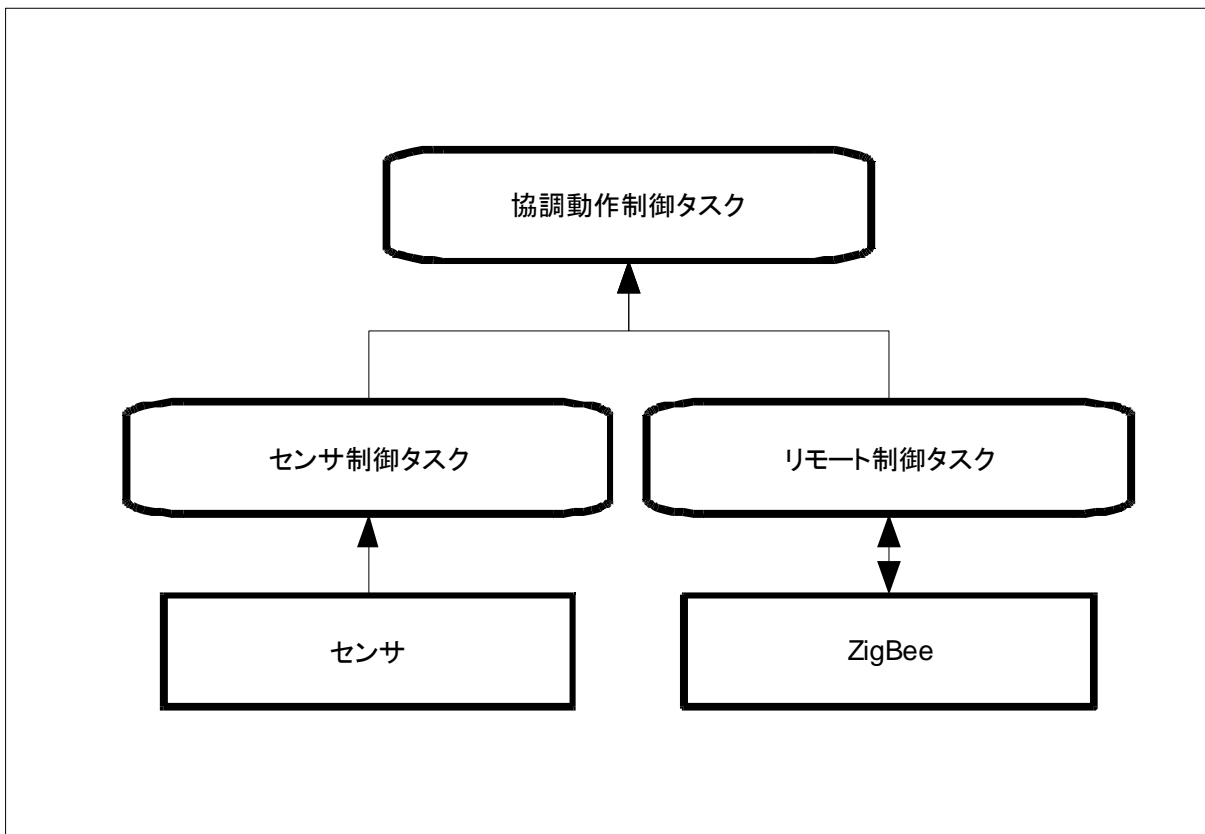


図 I-27-6. タスク関連図

## 【解説】

### 1) タスク関連図の作成

- \* 一般に、要件定義書を記述した後、外部設計書を記述する。この外部設計書の一部としてタスク関連図がしばしば利用される。
- \* タスク関連図はリアルタイムシステムの最適化に向いているとされ、詳細設計書への橋渡しとなる。
- \* 経済産業省の組み込み実態調査によると、米国の組み込み産業では 30%程度が利用しているとされる。
- \* タスク関連図に記載する項目は以下の通り
  - タスク番号と優先順位
  - タスクの起動関係
  - 起動タイミング
  - タスク間のキュー
  - タスク間の同期と排他制御
  - 重要なフラグの更新と参照の関連

スキル区分	OSS モデルカリキュラムの科目	レベル
組み込み分野	27 組み込みシステム最適化に関する知識 I	基本
習得ポイント	I-27-7. リアルタイム処理に対する評価項目と留意点	
対応する コースウェア	第 5 回（性能最適化の評価項目）	

## I-27-7. リアルタイム処理に対する評価項目と留意点

リアルタイムシステムの性能評価について、全体のパフォーマンス、処理のスループット、全体の優先順位、デバイスドライバの処理方法などの観点からみた評価項目と評価時の留意点を説明する。またタスク分割の設計評価についても解説する。

### 【学習の要点】

- \* リアルタイム処理の評価項目としてパフォーマンス、全体の優先順位、デバイスドライバの処理方式がある。
- \* タスク分割の設計評価として、キューの利用、排他制御、同期の利用に関する評価が考えられる。

パフォーマンス	<ul style="list-style-type: none"> <li>・コンテキストスイッチの頻発</li> <li>・カーネルのスケジューリング方式</li> <li>・スループットの計測</li> <li>・システムコールの計測</li> </ul>
全体の優先順位	<ul style="list-style-type: none"> <li>・リアルタイム性の重視、および分析手法の適用</li> </ul>
デバイスドライバの処理方式	<ul style="list-style-type: none"> <li>・ボトルネックの検出</li> </ul>

図 I-27-7. リアルタイム処理の評価項目

## 【解説】

### 1) 評価項目と留意点

- \* パフォーマンス
  - 頻繁なコンテキストスイッチによるパフォーマンスの低下が起こらないかタスクの切り替え頻度を確認する。
  - カーネルによる全体のスケジューリング方式と、アプリケーションの割り込みの頻度といった動作の特徴が合致しているかを確認する。
  - パフォーマンス指標の一つとしてスループットが用いられる。スループットは、システムへの入力に対して生成できる出力の比率と定義される。スループットにより、ハードウェアやソフトウェアを含むシステム全体のパフォーマンスを計量することができる。
  - システムコール単位でパフォーマンスを計測することもある。
- \* 全体の優先順位
  - プリエンプションの発生タイミング、処理の遅延などからタスクの優先度を決定し、リアルタイム性を優先した順位付けを行う。
- \* デバイスドライバの処理方式
  - デバイスドライバの動作手順に着目し、シミュレーションモデルや実際の計測によりパフォーマンスを評価し、避けられるボトルネックなどを特定する。例えば割り込みに与える優先度などを検討する。
  - デバイスドライバの動作手順は、処理要求待ち、入出力キューの作成、入出力の開始、割り込み処理に際してアプリケーションインターフェイスを起動、入出力の終了、と一般になっている。

### 2) タスク分割の設計評価

- \* スケジューラビリティによる分析が、タスクの設計評価の一つとして挙げられる。
- \* 使用するスケジューリングアルゴリズムに基づいて全てのタスクがリアルタイム性を維持し、かつ最適なプロセッサ利用率を達成できるかを分析する。
- \* レートモニタリング分析(Rate Monotonic Analysis: RMA)がリアルタイムシステムではよく使われる。
- \* また、キューイング、排他制御、同期に関してオーバーヘッドの大きさを確認することも行われる。

スキル区分	OSS モデルカリキュラムの科目	レベル
組み込み分野	27 組み込みシステム最適化に関する知識 I	基本
習得ポイント	I-27-8. CPU の性能指標と評価方法	
対応する コースウェア	第 6 回 (組み込みソフトウェアの概要)	

## I-27-8. CPU の性能指標と評価方法

CPI (Clock Cycles Per Instruction)、MIPS (Million Instructions Per Second)など、CPU の性能を評価する指標を紹介する。ただし様々なシステムの利用状況においては、これらの指標だけでは不十分であり個別の最適化手法を適用しなければならないことについても言及する。

### 【学習の要点】

- \* CPU の評価指標としては、CPU 時間(CPI)、単位時間当たりの実行可能命令数(MIPS)、単位時間当たりの実行可能演算数(MFLOPS)、などが使われる。
- \* SPEC(Standard Performance Evaluation Corporation)の作成する整数演算の性能評価用の SPECint などは CPU の性能評価ベンチマークとして使われている。
- \* 組み込みシステム向けのベンチマークとしては、EEMBC (EDN embedded microprocessor benchmark consortium)が有名である。

評価指標	CPI	1クロックあたりに実行可能な命令数
	MIPS	1秒間あたりの演算数(単位100万回)
	MFLOPS	単位時間あたりに実行可能な浮動小数点演算数(単位100万回)
ベンチマーク	SPEC	SPECint/SPECfpは整数/浮動小数点演算の性能を調べるベンチマーク
	EEMBC	組み込み機器向けのベンチマーク

図 I-27-8. CPU の評価手法とベンチマーク

## 【解説】

### 1) CPU の性能評価

- \* CPU の性能評価は 2) に示す指標を用いて計測される。

### 2) CPU の性能を示す指標

1 種類の測定法のみでは CPU 性能の 1 側面しか判断できないため、複数の指標が存在している。

- \* CPI (Clock cycles per instruction)
  - 1 クロックあたりに実行可能な命令数
- \* MIPS (Millions of instructions per second)
  - 1 秒間あたりの演算数(単位 100 万回)
- \* MFLOPS (Millions of floating-point operations per second)
  - 単位時間あたりに実行可能な浮動小数点演算数(単位 100 万回)

### 3) 標準ベンチマーク

- \* 客観的な統一指標で性能を比較する必要から、ベンチマークが作られている。
- \* 1980 年代に非営利法人の SPEC が作成する標準ベンチマークが有名である。
  - SPEC ベンチマークの結果は SPECmark と呼ばれ、ベンダ非依存の性能として示される。
  - SPECint/SPECfp  
整数/浮動小数点演算の性能を調べるベンチマーク
- \* 組み込み機器向けのベンチマークとして EEMBC (EDN embedded microprocessor benchmark consortium)がある。
  - Automotive, Consumer  
自動車、カメラ、携帯電話などの用途のマイクロプロセッサの性能評価
  - Java, Networking  
携帯電話や PDA 用の J2ME アプリケーションや、ルータなどのネットワーク製品用途のプロセッサの性能評価

### 4) CPU の性能に悪影響を及ぼすシステム

- \* 割り込みを多用するシステム
- \* 受付後に多数のレジスタの退避と復帰をハードウェアで行うシステム
- \* レジスタを命令で退避するシステム

スキル区分	OSS モデルカリキュラムの科目	レベル
組み込み分野	27 組み込みシステム最適化に関する知識 I	基本
習得ポイント	I-27-9. ソースコード最適化によるシステム性能向上	
対応する コースウェア	第 6 回 (ソフトウェアの最適化)	

## I-27-9. ソースコード最適化によるシステム性能向上

ソフトウェア最適化によるシステムの性能向上について、基本的な概念と設計方針を解説する。ソフトウェア最適化の基本手段である冗長なコードの排除やループの展開、インライン化、変数のレジスタ割り当てなど個別の手法を紹介し、またソースコードレベルの最適化の手順についても説明する。

### 【学習の要点】

- \* ソフトウェア最適化の方法としては、冗長なコードの削除、アルゴリズムの改善、ループ内不変値のくり出し、インライン化、などの手法が知られている。これらの手法により、メモリの利用率と、スピードが向上する。

冗長なコードの排除	実行効率に悪影響を及ぼしうる冗長なコードを排除することにより高速化ができる
ループ処理の最適化	ループ内の不変値のくりだし、ループのマージ、ループ回数の削減などの手法により高速化ができる
内部変数の最適化	プロセッサが高速に処理できるプリミティブを中心に使うことにより高速化ができる

図 I-27-9. ソースコード品質の改良によるソフトウェア最適化

## 【解説】

### (1)ソフトウェア最適化の方法

ソフトウェアの実行効率を最適化する手法として、言語仕様の活用、プロセッサ特性の活用、コンパイラの拡張機能・最適化機構の利用、などが挙げられる。他にもリンケージエディタによる最適化などが考えられる。

#### \* 冗長なコードの排除

- 冗長なコードは、仕様を変更する際に修正が複数個所に及び、手間がかかることに加え、実行効率に悪影響を及ぼしうる。

#### \* ループ処理の最適化

- ループ内不変値のくくりだし
- ループのマージ
- ループ回数の削減

#### \* 内部変数の最適化

- 関数内部で利用される内部変数(局所変数)で利用される型によって性能に大きな影響がある。
- プロセッサにとって高速に処理ができる型とそうでない型がある。
- 原始型(プリミティブ)の「整数型」「ポインタ型」は高速に処理できるが、「浮動小数点型」はFPU(Floating Point Unit)を搭載していないと高速には処理ができない。
- 集成型・派生型の実行は原始型に比べ遅くなる。

スキル区分	OSS モデルカリキュラムの科目	レベル
組み込み分野	27 組み込みシステム最適化に関する知識 I	基本
習得ポイント	I-27-10. プログラムモジュール配置による最適化	
対応する コースウェア	第 6 回 (ソフトウェアの最適化)	

## I-27-10. プログラムモジュール配置による最適化

プログラムモジュールの配置方法に係る最適化手法を解説する。プログラムモジュールのメモリへのローディング方法、リンク方式、常駐化など具体的な方法について説明する。

### 【学習の要点】

- \* 組み込みシステムの実行プログラムは単一リンクモデルとローディングモデルのどちらかを取る。
- \* 単一リンクモデルでは、すべてのプログラムをリンクによってひとつの実行可能形式プログラムとした後、ROM 化し、実行する。
- \* ローディングモデルは、外部記憶装置や ROM から実行に必要なデータを順次読みだすモデルである。

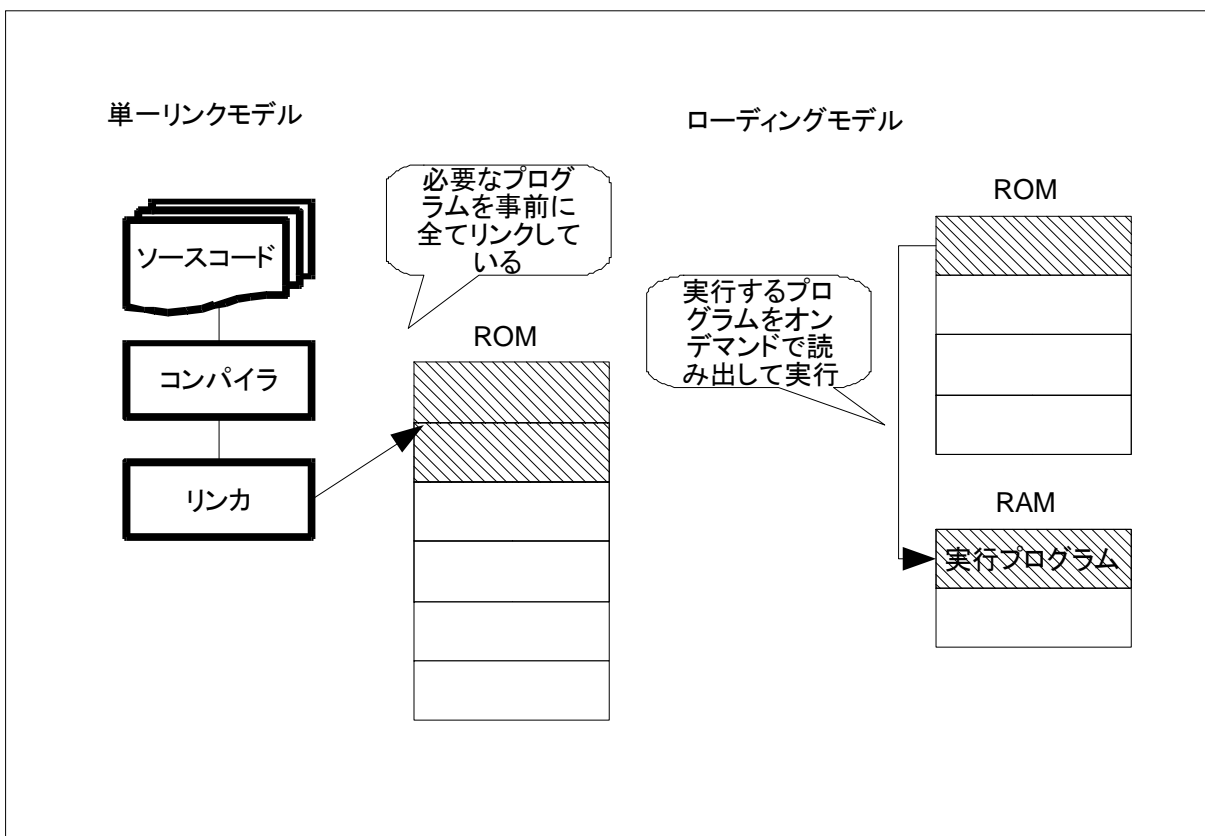


図 I-27-10. 単一リンクモデルとローディングモデル

## 【解説】

### 1) プログラムと処理

- \* プログラムのリンク方式について、組み込みシステムの実行プログラムは単一リンクモデルとローディングモデルのどちらかを取る。
  - 単一リンクモデルでは、すべてのプログラムをリンクによってひとつの実行可能形式プログラムとした後、ROM 化し実行する。
  - ローディングモデルは、外部記憶装置や ROM から実行に必要なデータを順次読みだすモデルである。
- \* 常駐と非常駐
  - プログラムを RAM 上に配置しておく(常駐)ことで、プログラムの起動を高速化することができる。しかし、RAM の容量は無限では無いため、使用頻度の少ないプログラムは非常駐になるように動作させるべきである。
- \* リューザブル化
  - リューザブル化とは、一度プログラムを実行した後に再度実行をしても正しく処理できるようにすることを指す。
  - メモリに常駐するプログラムはその都度ローディングされる訳ではなく、リューザブルを損なわない必要がある。この場合、初期値を書き換ええないなどの対応が必要になる。