

## 動的リコンフィギュラブルプロセッサへの 時間制約付き機能モジュール群分割アルゴリズムの検討

木谷 友哉<sup>†</sup> 中橋 亮<sup>††</sup> 中田 明夫<sup>††</sup> 安本 慶一<sup>†</sup> 東野 輝夫<sup>††</sup>

<sup>†</sup> 奈良先端科学技術大学院大学 情報科学研究科 〒630-0192 奈良県生駒市高山町 8916-5

<sup>††</sup> 大阪大学 大学院情報科学研究科 〒565-0871 大阪府吹田市山田丘 1-5

E-mail: <sup>†</sup>{t-kitani,yasumoto}@is.naist.jp, <sup>††</sup>{r-nakaha,nakata,higashino}@ist.osaka-u.ac.jp

あらまし 本稿では、マルチコンテキスト型動的再構成可能プロセッサへ時間制約を持つ実時間システムの機能モジュール群を分割し割り当てるためのアルゴリズムについて述べる。対象とするシステムを、時間制約を持つタスクグラフで表現し、それぞれのタスクをコンテキストに割り当てることでシステムの分割を実現する。本問題をコンテキストの最大サイズを最小化する整数線形計画問題 (ILP) として定式化を行った。その際、解の品質を落とすことなく ILP 問題にするための様々な制約を考案した。また、大規模なタスク割当問題に対処できるよう、二段階のヒューリスティックアルゴリズムを考案し、いくつかの例題に適用した結果、最適解の 1.1 から 1.3 倍程度のサイズの分割結果を比較的短時間で導出できることが分かった。

キーワード 実時間システム, 動的再構成可能プロセッサ, マルチコンテキスト, 整数線形計画法, スケジューリング

## A Context Assignment Algorithm for Functional Modules with Timing Constraints on Dynamic Reconfigurable Processor

Tomoya KITANI<sup>†</sup>, Ryo NAKAHASHI<sup>††</sup>, Akio NAKATA<sup>††</sup>,  
Keiichi YASUMOTO<sup>†</sup>, and Teruo HIGASHINO<sup>††</sup>

<sup>†</sup> Graduate School of Information Science, Nara Institute of Science and Technology  
8916-5 Takayama, Ikoma, Nara 630-0192, JAPAN

<sup>††</sup> Graduate School of Information Science and Technology, Osaka University  
1-5 Yamadaoka, Suita, Osaka 565-0871, JAPAN

E-mail: <sup>†</sup>{t-kitani,yasumoto}@is.naist.jp, <sup>††</sup>{r-nakaha,nakata,higashino}@ist.osaka-u.ac.jp

**Abstract** In this paper, we formally define a task decomposition problem for multiple functional modules with timing constraints on a multi-context dynamic reconfigurable processor. We model a real-time system as a set of task graphs with timing constraints, and decompose their tasks by assigning each task to a suitable context of the processor. We formulate the task decomposition problem as an integer linear programming problem. In order to treat large sized decomposition problems, we propose a heuristic algorithm. We show that the proposed heuristic algorithm derives quasi-optimal decomposition results for large sized examples in short time.

**Key words** real-time system, dynamic reconfigurable processor, multi-context, integer linear programming, scheduling

### 1. はじめに

近年のユビキタスネットワーク社会の発展に伴い、組み込みシステムは、短い開発サイクルで多品種少量生産することが求められている。そのため、小型化、省電力化を目的とした動的再構成可能プロセッサでの実現が注目されている。一般に、そのような組み込みシステムは、処理に時間制約のある実時間システムであり、動的再構成可能プロセッサ上で実現する場合、回路分割によって設計した回路が時間制約通りに動作しない可能性がある。そのようなシステムを実装する場合、あらかじめ時間制約を考慮して適切な回路分割法を考案するのが一般的である [1]。しかし、特定のアーキテクチャに依存した回路分割をそ

の都度考慮するのは煩雑であるため、システムの仕様からタスク群を、同時実行可能な部分タスク群に自動的に分割できることが望ましいが、このタスク分割問題は NP 困難な組合せ最適化問題である。従来、様々なタスク分割手法が研究されているが [2] [3]、これらの手法の多くは実時間制約を考慮しておらず、仕様の時間制約を満たすことは保証されていない。

本稿では、設計した実時間システムをマルチコンテキスト型の動的再構成可能プロセッサに実装する際に、与えられた時間制約を満たした上でシステムの実装に必要な論理領域のサイズが最小になるような処理タスク群の分割を求める手法を提案する。提案手法では、周期的な動作を行う並行システムを対象とする。各タスクには、処理の開始から終了までの時間や、処理

が必要とする論理構成領域の使用量(サイズ)、タスク間の半順序関係、分岐(融合)関係、実行開始時刻についての時間制約、などが指定できるものとする。

最適なタスク割当を求めるために、まず、上記の問題を整数線形計画問題(ILP)として定式化する。上記問題の制約条件は、コンテキストに割り当てられた各タスクが、それぞれの時間制約を満たした上で、あらかじめ指定された半順序に従い実行できるスケジューリング(各タスクの実行開始時刻)が存在することであり、上記問題の目的関数は、コンテキストのサイズの最大値の最小化である。提案する定式化では、対象とするシステムが持つ時間制約を表す制約式を全て整数線形不等式で表す。このとき、ILPにより効果的に解を求められるようにするため、対象システムが周期性を持つことに注目し、制約式に含まれる論理和を全て論理積の形で表せるようにした。

一般に解くべき問題のサイズが大きい場合、ILPを用いた上記の手法では、計算時間が大きくなるため、そのような問題に対して実行時間で解を得るためのヒューリスティックなタスク群分割アルゴリズムを提案する。これは二段階のアルゴリズムで構成され、第一段階では各タスクの実行開始時刻のスケジューリングを行い、第二段階では同一時刻に実行されるタスクの集合を最大コンテキスト数以下に抑えられるようにマージを行う。第一段階で、既存の最小リソース化スケジューリング Force-directed Scheduling [4] をベースに時間制約を満たすようにスケジューリングを行う。第二段階では、時間制約を満たすことを維持したまま、貪欲法にてタスクの集合をマージする。

実験を行った結果、複数の比較的小規模な例題に対してヒューリスティックで求めた解の評価値がILPによって求めた解の1.1~1.3倍程度に抑えられることを確認した。また、ILPでは実用的な時間で解が求まらない規模の例題に対しても、ヒューリスティックでは数秒のオーダーで解を求めることができた。さらに、ユビキタセンサを例に、システム設計における本手法の有用性を評価した。

## 2. 対象とするシステムのモデル化

### 2.1 実時間システム

#### 2.1.1 タスクグラフによる実時間システムのモデル化

実時間システムを3項組  $System = (Task, clock, N_t)$  で定義する。 $Task$  は実時間タスクの有限集合、 $clock$  は周期が1開始した時点からの経過時刻を表すカウンタ、 $N_t$  はシステムの周期を表す。 $Task$  の各要素  $\tau$  は、1つの変数  $t_{start}(\tau)$  と、6つの属性  $(T_{exe}(\tau), \tau_{prev}(\tau), guard(\tau), Task_{sync}(\tau), sync(\tau), size(\tau))$  を持つ。 $t_{start}(\tau)$  は、タスクの開始時刻を表す変数であり、 $clock$  の値が  $t_{start}(\tau)$  になったときに、 $\tau$  が実行される。 $T_{exe}(\tau)$  は、 $\tau$  の処理の実行時間を表す整数値である。ここで、タスクは処理を開始すると終了するまで処理を中断しないものとする。 $\tau_{prev}(\tau)$  は、フロー処理における  $\tau$  の先行タスクを表す。先行タスクの指定は、タスクをノードとして先行タスクから後続タスクに向かう有向枝を持つ有向グラフに閉路が生じないように行うものとする。 $guard(\tau)$  はタスクが満たすべき時間制約を表し、 $\tau$  の先行タスク群  $\tau'$  の  $t_{start}(\tau')$  を用いた線形式の論理積で表現される。 $Task_{sync}(\tau)$  は、 $\tau$  と同期して実行する可能性のあるタスクの有限集合である。 $sync(\tau)$  は  $\tau$  に対する同期ルールを表し、 $Task_{sync}(\tau)$  の要素を  $\wedge, \vee$  で結合した論理式で表す。 $\tau'$  が  $\tau$  と同期可能であるとき真となる論理変数を  $rendr(\tau)$  で表すとすると、 $\tau$  は  $sync(\tau)$  が真の時、同期可能である。例えば、

$$sync(\tau) = rendr(\tau') \wedge (rendr(\tau'') \vee rendr(\tau''')) \quad (1)$$

におけるタスク  $\tau$  は  $\tau'$  と同時に実行されるか、もしくは  $\tau'$ 、 $\tau''$  と同時に実行される。 $Task_{sync}(\tau) = \phi$  のとき、 $sync(\tau)$  は恒真である。 $size(\tau)$  は  $\tau$  をデバイス上に実装するために必要な構成領域(ロジックサイズ)を表す定数である。初期タスクから始まるタスクの処理系列は木状となり、これをプロセスと呼ぶ。システムは複数プロセスを並列に動作させる。また、 $clock = N_t$  になると全プロセスの処理は初期タスクに戻り、 $clock = 0$  となる。

#### 2.1.2 モデルの動作説明

図1にシステムのモデル例を示す。図中の実線矢印がタス

クを表しており各タスクの()内は  $T_{exe}(\tau), size(\tau)$ 、{}内は  $sync(\tau)$ 、[]内は  $guard(\tau)$  の情報である。

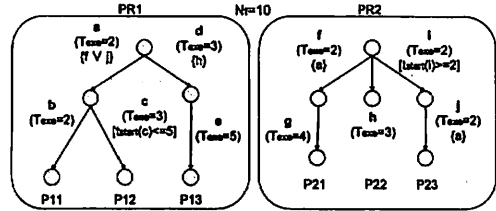


図1 実時間システムモデルの例

タスク  $\tau$  が実行を開始できる条件は以下が全て満たされたときである。

- 先行タスク  $\tau_{prev}(\tau)$  が終了
- 時間制約  $guard(\tau)$  が真
- $sync(\tau)$  が真

この条件を実行可能条件と呼ぶ。また、 $\tau$  が実行開始条件を満たしている間の時刻範囲を  $\tau$  の実行可能時間と呼ぶ。 $\tau$  は実行可能時間内の任意の時刻に実行が可能であるが、 $\tau$  の開始を遅らせることにより、後続タスクの実行可能時間を減少させる可能性がある。あるタスク  $\tau$  の実行可能時間が最も大きくなるのは、 $\tau$  より前に実行可能な全てのタスクが、開始条件を満たした次第即座に実行される場合である。このときの  $\tau$  の実行可能時間を  $\tau$  の最大実行可能時間と定義する。最大実行可能時間を持たないタスクは実行が不可能である。

システムの1周期に行われる動作はプロセスのパスの組合せで表される。しかし、同期するタスクの関係などにより、いくつかの組合せが実行不可能になる可能性がある。例えば図1に示す仕様からは同期関係のみを考慮すると図2のC1~C5までの5つの組合せを導出することができる。しかし、時間制約も考慮すると、系列C5において、タスク  $a$  の終了は最も早くても時刻6であるのでタスク  $c$  の時刻5以内に開始という制約を満たせない。よってC5は実行不可能な組合せとなり、C1~C4までが図1の仕様に対する実行可能な組合せであることが分かる。このように1回の周期で実行されるタスク系列の組合せを実行可能な実行系列と呼ぶ。実行系列  $p$  に含まれるタスクの集合を  $Task_{path}(p)$  と表す。

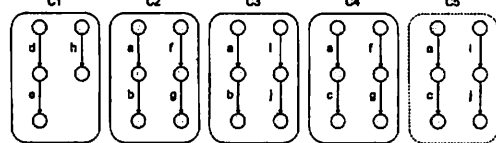


図2 実行可能な実行系列の導出例

### 2.2 動的再構成可能プロセッサ

現在、研究されている動的再構成可能プロセッサとして、NECエレクトロニクス社のDynamically Reconfigurable Processor(DRP) [5]、アイビーフレックス社のDAP/DNA-2 [6]等が存在する。DRPに代表されるマルチコンテキスト型動的再構成可能プロセッサは、数bitのレジスタやALUを持つPE(Processing Element)を回路の構成単位としたコンテキストと呼ばれる回路設定を複数保持した構造を持ち、制御信号を切り替えることでこれを1クロックで切り替えることが可能である。複数のコンテキスト間で同じ処理が実装されている場合、その処理はそのコンテキスト間の切り替えによって妨げられないものとする。

## 3. タスク割当問題

### 3.1 コンテキストへのタスク割当

システムを動的再構成可能プロセッサに実装するとき、分割した部分回路はそれぞれコンテキスト単位で実装される。そのため、システムの分割は各コンテキストに割り当てるタスク群を指定することで表現する。

### 3.1.1 タスク割当による最大実行可能時間の減少

システムのタスク群を分割してコンテキストに割り当てると、分割によっては各タスクの開始時刻が遅れが生じ、最大実行可能時間が小さくなる可能性がある。図 3(i) に示すような 2 つの並行に動作するタスク群を 2 枚のコンテキストに割り当てることを考える。ここで、各タスクは実行可能になると直ちに

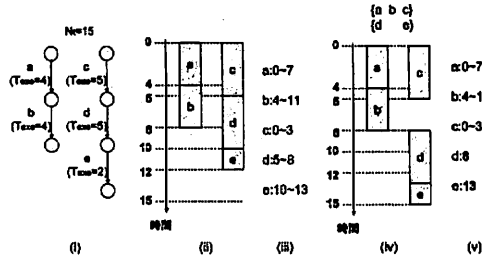


図 3 システム分割による最大実行可能時間の減少例

実行すると仮定したとき、このタスク群は同図 (ii) で表される時間範囲で動作する。各タスクの最大実行可能時間は同図 (iii) で表される範囲になる。このタスク群を 2 つのコンテキスト  $C_1 = \{a, b, c\}$ ,  $C_2 = \{d, e\}$  に分割して実装することを考えると、同じコンテキストに割り当てられていないタスク b と d は並行に動作することができず、タスク d の開始時刻が遅れが生じ、同図 (iv) で表される実行範囲で動作する。このときに各タスクの最大実行可能時間は同図 (v) で表される範囲に減少することになる。タスク割当問題では、システムの分割を行っても、各タスクが実行不可能にならないように分割を求めることが必要である。

本稿で対象とする動的再構成可能プロセッサは、1 クロックでコンテキストの変更が可能であり、切り替わる前後のコンテキストの同じ位置に同じタスクが実装されている場合は、コンテキストの変更に関わらずシームレスに実行の継続が可能である。例えば、図 3 に示すように、タスク群をコンテキストに  $C_1 = \{a, b, c\}$ ,  $C_2 = \{d, e\}$  とタスク b を重複させて割り当てて実装することで、時刻 5 でコンテキストが  $C_1$  から  $C_2$  に変更されても b はそのまま実行を続けることができ、同図 (ii) の時間範囲で動作可能である。本来は、タスクを実装するコンテキスト上の位置も考慮する必要があるが、本稿で提案する手法では簡単化のため、前後するコンテキストに重複して割り当てられたタスクが実行途中でコンテキストの変更が可能であると仮定する。このような複数のコンテキストに割り当てるタスク割当は最大実行可能時間の減少は少ないが、面積効率が悪くという問題がある。

### 3.2 タスク割当問題の制約

動的再構成可能プロセッサにおけるタスク割当問題は、以下の 2 つの制約を満たす必要がある。

- 全タスクは有効なスケジューリングを持つ
  - 全ての時刻において、各実行系列の並行に動作するタスク群が同時に実行可能となるようなコンテキストが存在する
- 以下では、それぞれについて説明する。

#### 3.2.1 タスクの有効なスケジューリング

各タスクの有効なスケジューリングとは、すべての時間制約を満たすように各タスクの実行開始時刻を決定することである。各タスクは 2.1.2 節で述べた実行開始条件を満たす必要がある。しかし、さらに下記条件を考慮する必要がある。

各タスクの実行開始時刻  $t_{start}(\tau)$  をただ 1 通りに求めることを考えると、実行可能時刻が異なる 2 つのタスクそれぞれと同期する場合などは、実際には実行可能なタスク割当が導出できるにもかかわらず解が求まらない。そのため、2.1.2 節で説明した各実行系列毎に、各タスクの開始時刻をスケジューリングする必要がある。これ以降、実行系列  $p$  のタスク  $\tau \in Task_{path}(p)$  の実行開始時刻を  $t_{p,\tau}$  と表す。

分岐のある実行系列において、タスクの実行開始時刻を求める場合、分岐に先行するタスクの実行開始時刻は、どの分岐に進んだ場合においても、それら後続タスクの時間制約を満たす

ようにスケジューリングされなければならない。具体的には、任意の系列  $p, p'$  において、共通の先行タスクの実行開始時刻は同一にスケジューリングする。例として、図 4 に示すシステムの実行系列  $p, p'$  を考える。このとき、それぞれの系列の共通タスクである a, b, g の実行中には、その後図中の網掛けされた後続タスクのどれか実行されるか、その時点では判断できない。タスク b に注目すると、次に c が実行されるか d が実行されるか判断できない。そのため、実行系列で独立してスケジューリングした場合に  $t_{p,b}$  と  $t_{p',b}$  が異なる値であるとする、遅い方の時刻にタスクを開始したときに共通でない後続タスクの時間制約が満たせなくなる可能性がある。そこで、任意の実行系列  $p, p'$  間での共通でない後続タスク群 (図中の網掛け部分) を差異タスク群と呼び、 $Task_{diff}(p, p')$  で表す。共通部分のタスク  $\tau \in Task_{path}(p) \cap Task_{path}(p')$  は、 $Task_{diff}(p, p')$  に含まれる全てのタスクの実行開始時刻よりも先に実行を開始する場合、その実行開始時刻は  $t_{p,\tau} = t_{p',\tau}$  を満たすという制約を与える。このような、分岐により分かれた実行系列間において、後続のタスク群の有効なスケジューリングを保証する条件を分岐実行系列間条件と呼ぶ。ここで、 $Task_{diff}(p, p')$  に含まれるタスクが 1 つでも  $\tau$  より先に実行開始される場合は、 $\tau$  が実行系列  $p$  として実行しているのか、 $p'$  として実行しているかが判断可能であるため、このタスクは分岐実行系列間条件を満たす必要はない。

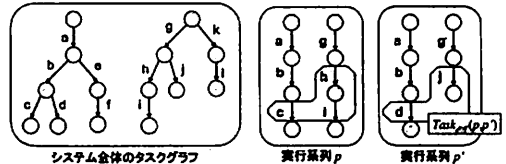


図 4 差異タスク群  $Task_{diff}(p, p')$

### 3.2.2 実行系列の実行可能性

動的再構成可能プロセッサ上で実行系列が実行可能であるかどうかは時間制約だけでなく、コンテキストへのタスク割当にも依存する。任意の実行系列  $p$  について、ある時刻  $t$  において、その時刻に実行中である系列内の全てのタスクを実装しているコンテキスト  $c$  が存在するとき、 $p$  は時刻  $t$  で動作可能であると呼ぶ。全タスクの有効なスケジューリングを求めたとしても、 $p$  が動作可能なコンテキストがない時刻が存在すれば、その系列は実行不可能である。

実行系列  $p$  が実行可能となる条件は、全ての時刻において  $p$  が動作可能であることである。

### 3.3 タスク割当問題の定式化

タスク割当問題を以下のように定式化する。

#### (a) 入力

- 実行時間システムのモデル  $System = (Task; clock; N_c)$
- 動的再構成可能プロセッサが保持するコンテキスト数  $N_c$

(b) 出力 タスク  $\tau \in Task$  がコンテキスト  $c$  ( $1 \leq c \leq N_c$ ) に割り当てられたとき 1 になるタスク割当 0-1 変数  $c_{c,\tau}$

(c) 制約条件  $System$  が分割前に実行可能である全実行系列は、分割後も実行可能

(d) 目的関数 コンテキストのサイズの最大値を最小化

$$\max_{1 \leq c \leq N_c} \left( \sum_{\tau \in Task} (c_{c,\tau} \cdot size(\tau)) \right) \rightarrow \min \quad (2)$$

### 4. 提案するタスク割当手法

#### 4.1 タスク割当問題の ILP 定式化

タスク割当問題の厳密な最適解を求めるために、3.3 節で示した問題を ILP を用いて定式化する。これはタスク割当問題の制約条件を全て整数線形不等式で表現し、その制約の下、最大のコンテキストサイズが最小になるように解くことで定式化する。

タスク割当問題の制約は、

- 全てのタスクが有効なスケジューリングを持つこと (3.2.1 節)

• 全ての実行系列が実行可能であること (3.2.2 節)  
である。以下では、ILP で問題を解くため、それらの制約条件を整数変数の線形不等式の論理積で表現する方法について、詳しく説明する。

#### 4.1.1 有効なスケジューリングを持つための制約式

タスクが有効なスケジューリングを持つとは、以下の 2 つの条件が満たされたときである。

- 実行開始条件 (2.1.2 節)
- 分岐実行系列間条件 (3.2.1 節)

それぞれについての ILP 制約式を以下に示す。

##### a) 実行開始条件を表す制約式

まず、タスク  $\tau$  において、先行タスク  $\tau' (= \tau_{prev}(\tau))$  が終了している条件を次式で表す。  $N_p$  は System の全実行系列数である。

$$\begin{aligned} \forall \tau (t_{p,\tau} \geq t_{p,\tau'} + T_{exe}(\tau')) \\ (1 \leq p \leq N_p, \tau \in Task_{path}(p)) \end{aligned} \quad (3)$$

次に、タスク  $\tau$  の時間制約  $guard(\tau)$  は、 $\tau$  の先行タスク以前のタスク  $\tau'$  の実行開始時刻を使った整数線形不等式の論理積であるため、 $guard(\tau)$  をそのまま ILP の制約式として用いる。

最後に、同期条件  $sync(\tau)$  は、同期するタスク  $\tau, \tau'$  間で実行開始時刻を同時にするため、次式のように表す。

$$\begin{aligned} t_{p,\tau} = t_{p,\tau'} \\ (1 \leq p \leq N_p, \tau \in Task_{path}(p), \\ \tau' \in Task_{path}(p), \tau' \in Task_{sync}(\tau)) \end{aligned} \quad (4)$$

なお、それぞれの実行系列内においての同期は全て and の同期となる (or の場合はそれぞれ別の実行系列に分かれる)。

##### b) 分岐実行系列間条件を表す制約式

任意の 2 つの実行系列間  $p, p'$  のタスク  $\tau (\in Task_{path}(p) \cap Task_{path}(p'))$  における分岐実行系列間条件は、以下の制約式で表すことができる。

$$\begin{aligned} \forall \tau' ((t_{p,\tau} < t_{q,\tau'}) \vee (t_{p',\tau} < t_{q,\tau'})) \rightarrow (t_{p,\tau} = t_{p',\tau}) \\ (\tau' \in Task_{diff}(p, p'), q \in \{p, p'\}, \tau' \in Task_{path}(q)) \end{aligned} \quad (5)$$

しかし、この式は整数線形不等式の論理積で表せていないため、以下の手順により等価な整数線形不等式を導出する。

まず、実行系列  $p, p'$  内のタスク  $\tau, \tau' (\in Task_{diff}(p, p'))$  において、 $\tau$  の実行開始時刻が  $\tau'$  の実行開始時刻より早く、 $\tau$  が分岐実行系列間条件を考慮する必要がある場合 1、そうでない場合 0 となる 0-1 変数  $pc_{p,p',\tau'}(\tau)$  を導入する。 $pc_{p,p',\tau'}(\tau)$  がそのような値を取るための制約式として、システムの周期が  $N_i$  であることを利用して以下の式を導入する。

$$\begin{aligned} t_{q,\tau} + N_i \geq t_{r,\tau'} + N_i \times (pc_{p,p',\tau'}(\tau)) \\ (1 \leq p, p' \leq N_p, \tau \in Task_{path}(p) \cap Task_{path}(p'), \\ q, r \in \{p, p'\}, \tau' \in Task_{diff}(p, p') \cap Task_{path}(r)) \end{aligned} \quad (6)$$

このとき一つでも  $t_{r,\tau'} > t_{q,\tau}$  であるような  $\tau'$  が存在する場合、 $pc_{p,p',\tau'}(\tau) = 0$  でなければ上記の制約式を満たさなため、この変数の定義を満たす。

変数  $pc_{p,p',\tau'}(\tau)$  を用いることで、式 (6) は次式 (7) に変換できる。(紙面の都合上、これ以降  $\sum_{\tau' \in Task_{diff}(p, p')} pc_{p,p',\tau'}(\tau)$  を  $PC_{p,p',\tau'}(\tau)$  と表す。)

$$(PC_{p,p',\tau'}(\tau) = 0) \rightarrow (t_{p,\tau} = t_{p',\tau}) \quad (7)$$

全ての  $p, p'$  で式 (7) が成り立つことは、以下の式 (8)(9) で表せる。

$$t_{p,\tau} + N_i \geq t_{p',\tau} + (1 - PC_{p,p',\tau'}(\tau)) \times N_i \quad (8)$$

$$t_{p,\tau} + (1 - PC_{p,p',\tau'}(\tau)) \times N_i \leq t_{p',\tau} + N_i \quad (9)$$

$$(1 \leq p, p' \leq N_p, \tau \in Task_{path}(p) \cap Task_{path}(p'))$$

式 (8)(9) は  $PC_{p,p',\tau'}(\tau) \geq 1$  とき、常に成り立つ。次に  $PC_{p,p',\tau'}(\tau) = 0$  のときを考える。このとき、式 (8)(9) は  $(t_{p,\tau} \geq t_{p',\tau}) \wedge (t_{p,\tau} \leq t_{p',\tau})$  と書き換えられるため、 $t_{p,\tau} = t_{p',\tau}$  を表す。以上により、分岐実行系列間条件を整数線形不等式とした制約式で表すことができる。

#### 4.1.2 実行系列が実行可能であるための制約式

実行系列  $p$  について、各時刻  $t$  に実行中のタスク  $\tau (\in Task_{path}(p))$  の集合をそれぞれ全て割り当てたコンテキスト  $c$  が存在するとき、 $p$  は動作可能である。ここでは全ての  $p$  が動作可能であることを表す制約式を、整数線形不等式で表すことについて説明する。

上記の条件を表現するために、実行系列  $p$  の各タスク  $\tau (\in Task_{path}(p))$  の実行開始時刻  $t_{p,\tau}$  において、その時刻で実行中の  $p$  のタスク群を全て割り当てたコンテキスト  $c$  が存在するときのみ 1 になる 0-1 変数  $cs_{c,p,\tau}$  を導入する。この変数を用いると上記の条件は次式で表すことができる。

$$\begin{aligned} \sum_{1 \leq c \leq N_c} cs_{c,p,\tau} \geq 1 \\ (1 \leq p \leq N_p, 0 \leq t < N_i, \tau \in Task_{path}(p)) \end{aligned} \quad (10)$$

ここで、各タスクの実行開始時刻  $t_{p,\tau}$  のみにおいて、条件を満たすコンテキスト  $c$  を求めることができれば、それ以降、他のタスクが新たに実行を開始するまで、同時に動作するタスクの集合の要素が増えることはないため、その  $c$  によって  $p$  は動作可能であることが判断できる。

以降、この変数  $cs_{c,p,\tau}$  が目的通りの値を取るための制約式について考える。 $cs_{c,p,\tau}$  に関する制約式が他にない場合、ILP ソルバでは、式 (10) を満たすために  $cs_{c,p,\tau} = 1$  となるように変数値の設定が行われる。そのため、実行系列  $p$  のタスク  $\tau$  がコンテキスト  $c$  で実行不可能な場合は必ず  $cs_{c,p,\tau} = 0$  にする制約式を導出する。 $cs_{c,p,\tau} = 0$  となる条件は以下の通りである。

- コンテキスト  $c$  が、実行系列  $p$  中のタスク  $\tau (\in Task_{path}(p))$  を割り当てていない。
  - コンテキスト  $c$  が、同時に実行する実行系列  $p$  中のタスク  $\tau (\in Task_{path}(p))$  の実行開始時刻に動作中の他のタスクを全て割り当てていない。
- 前者の条件は、コンテキスト  $c$  にタスク  $\tau$  が割り当てていることを示すコンテキスト割当 0-1 変数  $c_{c,\tau}$  を用いて、次のように表すことができる。

$$\begin{aligned} c_{c,\tau} \geq cs_{c,p,\tau} \\ (1 \leq c \leq N_c, 1 \leq p \leq N_p, \tau \in Task_{path}(p)) \end{aligned} \quad (11)$$

次に、実行系列  $p$  内のタスク  $\tau'$  がタスク  $\tau$  の開始時刻に動作していることは  $t_{p,\tau'} \leq t_{p,\tau} < t_{p,\tau'} + T_{exe}(\tau')$  で表せるため、後者の条件は次式で表せる。

$$\begin{aligned} ((t_{p,\tau'} \leq t_{p,\tau} < t_{p,\tau'} + T_{exe}(\tau')) \wedge (c_{p,\tau'} = 0)) \\ \rightarrow (cs_{c,p,\tau} = 0) \\ (1 \leq c \leq N_c, 1 \leq p \leq N_p, \tau, \tau' \in Task_{path}(p)) \end{aligned} \quad (12)$$

式 (13) を整数線形不等式の形の制約式で表現するために、実行系列  $p$  においてタスク  $\tau$  とタスク  $\tau'$  の開始時刻を比較する 0-1 変数  $tc_{p,\tau,\tau'}$  を導入する。 $tc_{p,\tau,\tau'}$  は  $t_{p,\tau} \geq t_{p,\tau'}$  のとき 0、 $t_{p,\tau} < t_{p,\tau'}$  のとき 1 と定義する。変数  $tc_{p,\tau,\tau'}$  を用いることで、実行系列が実行可能である制約式である式 (13) は次のような整数線形不等式で表すことができる。

$$\begin{aligned} t_{p,\tau'} + T_{exe}(\tau') \leq t_{p,\tau} + N_i \times (c_{p,\tau} + (1 - cs_{c,p,\tau}) + tc_{p,\tau,\tau'}) \\ (1 \leq p \leq N_p, \tau \in Task_{path}(p), \tau' \in Task_{path}(p)) \end{aligned} \quad (13)$$

この式は  $tc_{p,\tau,\tau'} = 0$  (つまり  $t_{p,\tau'} \leq t_{p,\tau}$ ) かつ  $t_{p,\tau} < t_{p,\tau'} + T_{exe}(\tau')$  かつ  $c_{p,\tau} = 0$  かつ  $cs_{c,p,\tau} = 1$  の場合のみ偽になり、式 (13) と等価である。

最後に、導入した変数  $tc_{p,\tau,\tau'}$  が目的の値を取るための制約式を以下に示す。

$$t_{p,\tau'} \leq t_{p,\tau} + N_t \times tc_{p,\tau,\tau'} \quad (14)$$

$$t_{p,\tau'} - 1 + N_t < t_{p,\tau} + N_t \times tc_{p,\tau,\tau'} \\ (1 \leq p \leq N_p, \tau \in Task_{path}(p), \tau' \in Task_{path}(p)) \quad (15)$$

以上により、全ての制約条件を、整数線形不等式の論理積で表現することができ、ILP として解くことが可能である。

#### 4.2 ヒューリスティックアルゴリズム

ILP による解法はタスクグラフのサイズや、割り当てるコンテキストの枚数などにより計算時間が指数的に増加する。そこで、大きなサイズの問題を解くために、ヒューリスティックなアルゴリズムを考案する。タスク割当問題の許容解の条件は、その実行系列に含まれる全タスクが有効なスケジューリングの全時刻に対して動作可能なコンテキストを持つことの2つである。よって、以下のような二段階のアルゴリズムを考案した。第一段階では有効なスケジューリングをヒューリスティックに導出する。これは、ハードウェアの必要なリソース量を最小にするグリーディな既存のスケジューリングアルゴリズム Force-Directed Scheduling [4] をベースとした。第二段階ではそのスケジューリングで同時に動いているタスク集合を統合していくことにより、タスク割当問題の解を導出する。

##### 4.2.1 Force-Directed Scheduling

Force-Directed Scheduling (FDS) [4] は、与えられた各処理に割り当てる演算器リソースの量を回路の各動作時刻において平均化し、リソース使用量の最大値を最小化するヒューリスティックなスケジューリング手法である。各タスクは、各タスクの動作順序の前後関係や、実行所要時間、デッドラインから実行可能となる時刻の範囲を持つ。この時刻範囲にタスクが等確率で割り当てられた場合の必要リソースの期待値を評価関数に利用することでリソース使用量の平均化を図っている。ここで、 $\tau$  の実行可能時間のうち最も早い時刻を最早実行開始可能時刻  $T_{est}(\tau)$ 、最も遅い時刻を最遅実行開始可能時刻  $T_{lat}(\tau)$  と呼ぶとき、時刻  $t$  ( $T_{est}(\tau) \leq t \leq T_{lat}(\tau) + T_{exe}(\tau) - 1$ ) においてタスク  $\tau$  が使用するリソースの期待値  $EXP_t(\tau)$  は

$$EXP_t(\tau) = \frac{\{\min\{t - T_{est}(\tau) + 1, T_{lat}(\tau) - T_{est}(\tau) + 1, T_{lat}(\tau) + T_{exe}(\tau) - t + 1\}}{\text{size}(\tau)} \times \frac{\text{size}(\tau)}{T_{lat}(\tau) - T_{est}(\tau) + 1} \quad (16)$$

で表される。 $t < T_{est}(\tau)$ 、 $T_{lat}(\tau) + T_{exe}(\tau) \leq t$  の時刻では  $EXP_t(\tau) = 0$  である。同時刻に割り当てられるタスクの合計サイズの最大値を最小にするように最も期待値の小さい時刻へ順に各タスクの実行開始時刻をスケジューリングしていく。図5にFDSの一例を示す。

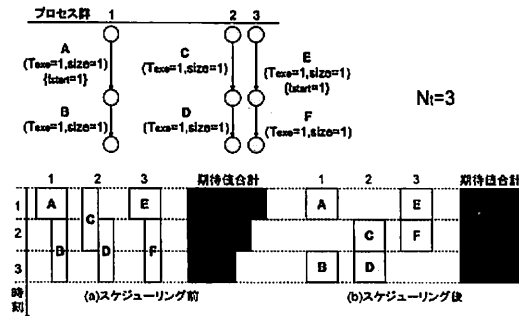


図5 FDSの例

##### 4.2.2 アルゴリズムの概要

各実行系列に対して、FDSを用いてスケジューリングを行う。ここで、分岐実行系列間制約を考慮するため、実行系列  $p$  で  $\tau$  が、 $Task_{diff}(p, p')$  に含まれる全てのタスク  $\tau'$  の最遅実行開始時刻  $T_{lat}(\tau')$  より前にスケジューリングされたとき、同

時に実行系列  $p'$  においても同時刻にスケジューリングする。このように各実行系列のスケジューリングを並行して行う。その時点で、最大リソースが大きい系列を選択し、一つタスクをスケジューリングし、再び系列の選択に戻ることを繰り返す。

このスケジューリングをもとに、タスク割当を導出する。どのタスクの開始時間でも、コンテキストが実行可能であるという制約を満たすため、同時に実行するタスク群を集合として抽出し、その集合群を  $N_c$  個に統合するというアプローチをとる。このとき、統合されたそれぞれのコンテキストにおいて、割り当てられたタスクの合計サイズの最大値が最小になるようにグリーディに統合を行う。

## 5. 評価実験

### 5.1 アルゴリズムの実験的評価

提案手法の有用性を確かめるため、ランダムに生成したタスクグラフに対し、ILP とヒューリスティックのそれぞれの手法を適用した。ILP ソルバとして GLPK [7] を使用し、ヒューリスティックな手法のプログラムは Java 言語 (JDK5.0) で作成した。評価環境は CPU: Pentium4 2.8GHz、メモリ: 2GB、OS: Windows XP Professional の計算機を用いた。

まず、各手法の計算時間を比較するため、タスク数を増加させた時の各手法の計算時間を比較した。 $N_c = 4$  とした時の結果が表1である。表の  $n \times n$  とは、 $n$  個のタスクから構成されるプロセスが  $n$  個並列に動作している例であることを示す。

表1 各手法の計算速度の違い

|           | 2×2  | 3×3  | 4×4   | 5×5  |
|-----------|------|------|-------|------|
| ILP       | 0.1s | 250s | 45min | >2h  |
| ヒューリスティック | 31ms | 47ms | 63ms  | 97ms |

この結果から、ヒューリスティックを用いた手法は ILP を用いた手法より計算時間が大幅に少ないこと、および、ILP はタスクの増加に伴い大幅に計算時間が増えているのに対し、ヒューリスティックは計算時間の増加が少ないことが分かる。

次にヒューリスティックの解の精度を比較するため、時間制約が緩い例題と時間制約が厳しい例題それぞれに対し10個のランダムなグラフを用意した。時間制約が緩い例題の全てのタスクは最遅実行開始時刻が最早実行開始時刻の1.5倍以上であるようにグラフを作成し、時間制約が厳しい例題はわずかも遅れと実行不可能になるタスクが1つは存在するように作成した。ランダムグラフのその他のパラメータは、プロセス数3、1プロセスの平均タスク数3、分岐確率20%、平均ロジックサイズは5である。これらの例に対して  $N_c = 3$  として分割した結果を表2に示す。

表2 各手法の解の精度の違い

|          | 手法        | 最大サイズ | 実行時間   |
|----------|-----------|-------|--------|
| 時間制約が緩い  | ILP       | 16.2  | 207s   |
|          | ヒューリスティック | 17.8  | 37.8ms |
| 時間制約が厳しい | ILP       | 15.9  | 15.5s  |
|          | ヒューリスティック | 19.8  | 31.5ms |

時間制約が緩い例題については、最適解の約1.1倍のサイズの解を導出している。時間制約が厳しい例題については、最適解の約1.3倍のサイズの解になってしまっている。しかし、時間制約が厳しい例題に対しては ILP の解探索空間が小さくなり、ILP の計算時間が減少している。このことから、比較的小さな問題に対しては時間制約によって ILP とヒューリスティックを使い分けることによって精度の良い解が得られると考えられる。

### 5.2 システム設計における本手法の適用

本手法を図6のような機能の書き換えを想定した複数のセンサを使い分けるユビキタセンサの動作仕様に対して適用した。ユビキタセンサは図7のようなアーキテクチャを想定し、信号処理部(斜線部)は図8でモデル化される動作を行う。動作指示処理と同期して、センシング、通信が行われる。これらの

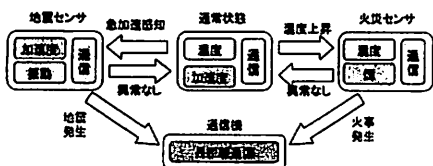


図6 ユビキタスセンサ動作イメージ

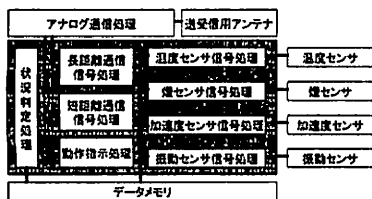


図7 ユビキタスセンサのアーキテクチャ

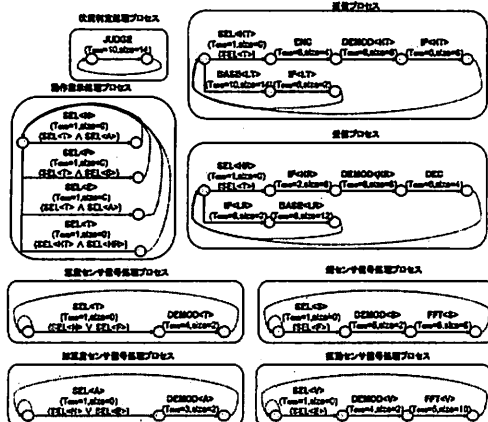


図8 ユビキタスセンサのモデル

通信処理はバスを利用した処理とし、ロジックサイズを0とおく、各センサは取得した情報の復調を行う。ただし、短センサと振動センサはそのあとにFFTによるデータ解析を行う。通信処理はベースバンド処理とIF処理を行うとし、長距離通信のベースバンド処理はエンコード/デコード処理と復調処理が順次行われるとする。これらの1周期前のデータを元に状況判定処理を行い、そのデータはその1周期後の動作指示処理に用いられる。周期間のデータのやりとりはデータメモリを使って行い、データメモリには適宜アクセスできるとする。各処理のロジックサイズ、実行時間はAnalog Devices社のセンサデータ及びAltera社が提供しているIPコアを参考にして決定した。

#### 5.2.1 アルゴリズムの実行結果

5.2節で作成したモデルに対して、ヒューリスティックアルゴリズムを使用してシステム分割を求めた。 $N_t$ は50、 $N_c$ は4とした。コンテキストにはサイズの存在するタスクの割り当てのみを考える。この結果、図9に示すシステム分割結果を約0.2秒の計算時間で導出した。最大のコンテキスト(太線)のサイズは36である。表3で示されるサイズ、コストの異なる3種類の動的再構成可能プロセッサが存在するとしたとき、このユ

表3 利用可能な動的再構成可能プロセッサ

|   | サイズ | コスト |
|---|-----|-----|
| A | 20  | 100 |
| B | 40  | 250 |
| C | 60  | 700 |

ビキタスセンサは動的再構成可能プロセッサBで実装可能であることが分かる。次に、高性能なセンシング処理のため、センサ信号処理のロジックサイズおよび実行時間が2倍になったモデルを考える。このときのシステム分割結果は図10のようになり、計算時間は約0.2秒、最大のコンテキストのサイズは48であった。この分割結果より高性能のユビキタスセンサは動的再構成可能プロセッサCを使用する必要があることが分かる。

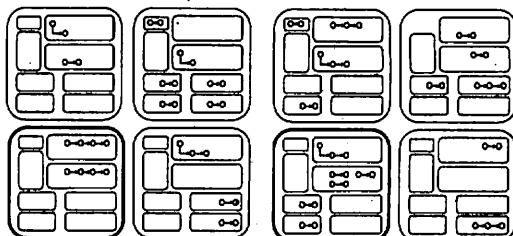


図9 ユビキタスセンサの分割結果 図10 高性能ユビキタスセンサの分割結果

このように、本手法を用いることで、その仕様において最も低コストである動的再構成可能プロセッサを選択できる。

## 6. おわりに

本稿では、実時間システムをマルチコンテキスト型動的再構成可能プロセッサに実装する際に、システムが与えられた時間制約を満たした上で実装に必要な論理領域のサイズが最小になるような実時間処理群の分割を求める手法を考案した。

対象とするシステムは、システムの各モジュールの処理をタスクとみなしたタスクグラフでモデル化し、タスク割当て時間制約を満たす制約を整数線形計画問題(ILP)に定式化した。次にタスクのスケジューリングとタスク割当てを二段階で行うヒューリスティックアルゴリズムを考案し、解の最大サイズが最適解の1.1~1.3倍程度に収まること、及び実用的な例題における本手法の有用性を確認した。

今後の課題としては、タスクの配置位置や、再構成時の外部メモリへのアクセスなどを考慮した手法の考案などが挙げられる。

## 文 献

- [1] M. Suzuki, Y. Hasegawa, Y. Yamada, N. Kaneko, K. Deguchi, H. Amano, K. Anjo, M. Motomura, K. Wakabayashi, T. Tui, and T. Awashima: "Stream Applications on the Dynamically Reconfigurable Processor," *Proc. of 2004 Int'l Conf. on Field Programmable Technology (FPT2004)*, pp. 575-580, 2004.
- [2] R. Maestre, R. Hermida, F.J. Kurdahi, N. Bagherzadeh, and H. Singh: "Optimal vs. Heuristic Approaches to Context Scheduling for Multi-Context Reconfigurable Architectures," *Proc. of 2000 IEEE Int'l Conf. on Computer Design (ICCD'00)*, pp. 575-576, 2000.
- [3] I. Taniguchi, K. Ueda, K. Sakanushi, Y. Takeuchi, and M. Imai: "Task Partitioning Oriented Architecture Exploration Method for Dynamic Reconfigurable Architectures," *Proc. of the 14th IFIP Int'l Conf. on Very Large Scale Integration (VLSI-SoC 2006)*, pp.290-295, 2006.
- [4] P.G. Paulin, and J.P. Knight: "Force-Directed Scheduling for the Behavioral Synthesis of ASIC's," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 8, No. 6, pp. 661-679, 1989.
- [5] NEC Electronics Corp.: Dynamically Reconfigurable Processors, <http://www.necel.com/ja/techhighlights/drpf/>.
- [6] IPflex Inc.: DAP/DNA-2, <http://www.ipflex.com/jp/>.
- [7] The GNU Operating System, GLPK, <http://www.gnu.org/software/glpk/glpk.html>.