社団法人 情報処理学会 研究報告
IPSJ SIG Technical Reports

2008－SLDM－134／2008－EMB－8 (31)
2008 / 3 / 28

再構成オーバーヘッドを考慮した動的再構成可能プロセッサへの実時間タスク群分割実装アルゴリズムの提案

西　圭祐† 木谷 友哉†† 中田 明夫††† 東野 輝夫†

† 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
†† 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
††† 広島市立大学 大学院情報科学研究科 〒731-3194 広島市安佐南区大塚東三丁目 4-1
E-mail: †{k-nisi,higashino}@ist.osaka-u.ac.jp, ††t-kitani@is.naist.jp, †††nakata@hiroshima-cu.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に実時間タスク群を実装するための，資源割付アルゴリズムを提案する．対象とするプロセッサは，一般にコンテキスト切り替え時にオーバーヘッドを有している．このため，これらのオーバーヘッドを考慮し，各タスクに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にタスクをコンテキストに割り付ける必要がある．そこで，この問題を ILP で定式化し，この問題を実用時間で解くためのヒューリスティックアルゴリズムを考案した．

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 Real-time Tasks on Dynamically Reconfigurable Processor with Reconfigurable Overhead

Keisuke NISHI†, Tomoya KITANI††, Akio NAKATA†††, and Teruo HIGASHINO†

† Graduate School of Information Science and Technology, Osaka University
1-5 Yamadaoka, Suita, Osaka 565-0871, JAPAN
†† Graduate School of Information Science,Nara Institute of Science and Techonology
8916-5 Takayama, Ikoma, Nara 630-0192, JAPAN
††† Graduate School of Information Science and Technology, Hiroshima City University
3-4-1 Ohtsukahigashi, Asaminami, Hiroshima, 731-3194, JAPAN
E-mail: †{k-nisi,higashino}@ist.osaka-u.ac.jp, ††t-kitani@is.naist.jp, †††nakata@hiroshima-cu.ac.jp

Abstract In this paper, we formulate a task assignment problem into a multi-context dynamic reconfigurable processor where real-time tasks with timing constraints are executed. We model a reconfigurable processor with reconfigurable overhead and the task assignment problem is formulated for the model. Our goal is we decompose a given task graph with timing constraints by assigning each task to a suitable context of the processor. We formulate the task assignment problem as an integer linear programming problem, and propose a heuristic algorithm to solve it in short time.

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

1. はじめに

ネットワークのユビキタス化に伴い，多種多様な組み込みシステムの開発・生産が行われている．そのような組み込みシステムは製造コストの観点から，再構成可能論理プロセッサに実装されることが多い．近年，そのプロセッサの論理構成領域の面積利用効率を改善するため，システムの動作中に動的に回路構成を変更できるマルチコンテキスト型動的再構成可能プロセッサの研究が盛んである．マルチコンテキスト型動的再構成可能プロセッサ (DRP) に実時間並行システムを実装する場合，システムを複数のタスクに分割し，各タスクの時間制約を考慮して複数のコンテキスト上に実装する．得られたコンテキストをあらかじめ定めたタイミングで切り替えていくことにより，システム全体の動作が実現される．これまでに，DRP に対する様々なタスク分割手法が研究されているが [1] [2] [3]，これらの手法の多くは組み込みシステムを構成する内部のタスクが満

たすべき実時間制約の存在を仮定しておらず，仕様が時間制約を満たすことは保証されない．本研究では，先行研究 [4] を拡張し，コンテキストの切り替え時のオーバーヘッドを考慮しつつ最適なタスク割り当てを求めるための問題を整数線形問題 (ILP) として定式化した．また，サイズの大きな問題に対しても実用的な時間で解を求められるよう，FDS (Force-directed Scheduling) と貪欲法を利用した 2 段階のヒューリスティックアルゴリズムを提案している．

提案手法では，コンテキスト切り替え時のオーバーヘッドやコンテキストの枚数などをパラメータとして取り扱うことができるようにすることで，より汎用性の高い動的再構成可能プロセッサを対象としたタスク割り当てが行えるようにしている．加えて，解くべき問題のサイズが大きい場合に対しても，実用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している．提案アルゴリズムでは，並行に動作する実行系列の各タスクをノードで表現し，同時にシステム上で実行される可能性があるタスク群に対応するノード間にエッジを引いたグラフを作る．このとき，サイズが最大のクリークを同じコンテキストに割り当て，1 つのノードとしてマージしていくことで，ある時刻において同時に実行される可能性があるタスクの集合の最大サイズを小さくする工夫を行っている．

予備実験で，複数の比較的小規模な例題に対して，提案するヒューリスティックで求めた最適解の評価値が ILP によって求めた解の 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で解を求めることができることを確認した．さらに，簡単な携帯電話端末の例題を元に，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

本稿で対象とする実時間システムはシステムの各処理に実時間制約がついたシステムであり，各処理は実時間タスクと呼ばれる．

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 τ は 6 つの属性を持ち，$< T_{exe}(\tau), \tau_{prev}(\tau), guard(\tau), Task_{sync}(\tau), sync(\tau), size(\tau) >$ で定義する．$T_{exe}(\tau)$ は τ が処理を開始してから処理を終了するまでに要する時間を表す整数定数である．$\tau_{prev}(\tau)$ は τ のフロー処理における先行タスクを表す．$guard(\tau)$ はタスクが満たすべき時間制約を表す．タスク τ の開始時刻，つまり τ の開始時の $clock$ の値を $t_{start}(\tau)$ とすると，$guard(\tau)$ は $t_{start}(\tau)$ を用いた線形式の論理積で表現される．$Task_{sync}(\tau)$ は τ と同期する可能性のあるタスクの有限集合である．$sync(\tau)$ は τ に対する同期ルールを表し，$Task_{sync}(\tau)$ の要素を and，or で結合した論理式で表す．$size(\tau)$ は τ 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) を表す定数変数である．

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．システムはプロセスの並列動作である．なお，対象とするシステムは周期的なシステムとし，$clock = N_t$ になると各プロセスの処理は初期タスクに戻り，$clock = 0$ になる．

2.1.2 モデルの動作説明

タスク τ 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
- 同期タスク $Task_{sync}(\tau)$ の中で現在実行可能なタスクを真とした時，$sync(\tau)$ が真

システムの 1 周期に行われる動作はプロセスのパス（タスクシーケンス）の組合せで表される．このパスの組合せを以降実行系列と呼ぶ．同期するタスクの関係などにより，いくつかの実行系列は実行不可能になる可能性がある．例えば図 1 で示す仕様からは時間制約を考慮せず同期関係のみを考慮すると，$(P11, P21)$, $(P11, P23)$, $(P12, P21)$, $(P12, P23)$, $(P13, P22)$ の 5 つの実行系列を導出できる．しかし，時間制約を考慮すると，$(P13, P23)$ において，タスク a の終了は i と j の終了を待つため，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 $(P13, P23)$ は実行不可能であり，それ以外の 4 つの実行系列が図 1 のシステムにおいて実行可能となる．

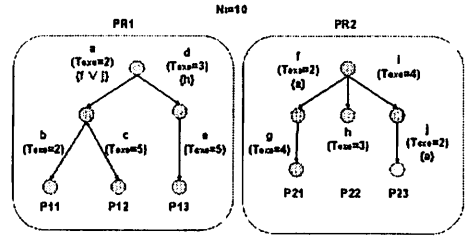

図 1 実時間システムの例

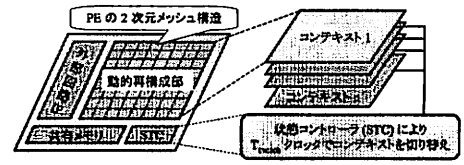

図 2 動的再構成可能プロセッサのアーキテクチャ

2.2 動的再構成可能プロセッサ

現在，マルチコンテキスト型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) [6]，アイピーフレックス社の DAP/DNA-2 [7] 等が考案されている．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コンテキストを数クロック程度のオーバーヘッドで切り替えることが可能である．

本稿で対象とするデバイスとして，図 2 に示すような構造を持つ一般的なマルチコンテキスト型の動的再構成可能プロセッサを想定する．本稿では，このような動的再構成可能プロセッサを $DRP = (N_c, T_{switch})$ で定義する．ここで，N_c はコンテキスト枚数，T_{switch} はコンテキスト切り替え時のオーバーヘッド時間を表す．

3. タスク割り当て問題

3.1 考慮する再構成オーバーヘッド

コンテキストに割り当てられたタスクは，以下のような場合，実行時にコンテキスト切り替えに要するオーバーヘッドを考慮しなければならない．

(I) 先行タスク τ' と後続タスク τ が違うコンテキストに割り当てられている

(II) 1 つのタスク τ が複数のコンテキストに割り当てられ，コンテキスト切り替え前後で動作している

(I) の場合，τ は τ' の実行後，少なくとも T_{switch} 待った後に実行が開始されなければならない．(II) では，複数のコンテキストをまたぐタスク τ は，実行中にコンテキストが切り替わった回数分の T_{switch} を実行時間として考慮しなければならない．

3.2 タスク割り当て問題の制約

動的再構成可能プロセッサにおけるタスク割り当て問題は，3.1 節で示した再構成オーバーヘッドを考慮した上で，以下の 2 つの制約を満たす必要がある [4].

(i) 全タスクは有効なスケジューリングを持つ

(ii) 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

3.3 タスク割り当て問題の定式化

タスク割り当て問題を以下のように定式化する.

(a) 入力

- 実時間システムのモデル $System = (Task, clock, N_t)$
- 動的再構成可能プロセッサのモデル $DRP = (N_c, T_{switch})$

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

(c) 制約条件 $System$ が分割によって実行可能性を失わない

(d) 目的関数 次式のコンテキストのサイズの最大値を最小化する

$$\max_{(1 \leqq c \leqq N_c)} \Big(\sum_{\tau \in Task} (c_{c,\tau} \cdot size(\tau)) \Big) \rightarrow min$$

4. 再構成オーバーヘッドを考慮したタスク割り当て問題の ILP 定式化

4.1 再構成オーバーヘッドに無関係な線形制約式 [4]

2.1.2 項で述べたように，システムは各周期においていずれかの実行可能な実行系列を実行する．実行系列はパスの組合せで構成されるため，異なる実行系列に同じパスが含まれ，その結果，同じタスクが複数の実行系列に含まれる．このとき，同じタスクであっても実行系列が異なれば，他のタスクの同期関係などによって，実行を開始できる時刻が異なる可能性がある．そのため，以下の変数を導入した.

- 整数変数 $t_{p,\tau}$: 実行系列 p 内のタスク τ に対して，そのタスクが p 内で実行を開始する時刻を表す

3.2 節で示した制約 (i) の有効なスケジューリングとは，全ての実行系列の全てのタスクに対して，時間制約を満たすように $t_{p,\tau}$ を決めることである.

次に，3.2 節で示した制約 (ii) を満たすことを示すための変数として以下の変数を導入した.

- 0-1 整数変数 $cs_{c,p,\tau}$: 実行系列 p 内のタスク τ が実行を開始したとき（時刻 $t_{p,\tau}$），実行を開始しているタスク群（$(t_{p,\tau'} \leqq t_{p,\tau}) \wedge (t_{p,\tau} < t_{p,\tau'} + T_{exe}(\tau'))$ であるような p 内のタスク τ' の集合）がコンテキスト c に割り当てられているなら 1，それ以外なら 0

また，タスクがコンテキストに割り当てられているかどうかを表すための変数として以下を導入した.

- 0-1 整数変数 $c_{c,\tau}$: タスク τ がコンテキスト c に割り当てられているなら 1，それ以外なら 0

再構成オーバーヘッドを考慮しないタスク割り当て問題の線形制約式については，紙面の都合上割愛する．詳しくは，文献 [4] を参照されたい.

4.2 再構成オーバーヘッドを考慮した線形制約式

再構成オーバーヘッドを考慮するため，コンテキスト切り替え時のオーバーヘッドを表す変数 T_{switch} を用いて線形制約式を定義する．T_{switch} を要するのは，3.1 節で示した (I) (II) の場合である．(II) では，あるタスク τ の実行中にコンテキストが切り替えられて実行が続けられとき，コンテキストが切り替えられた回数分のオーバーヘッド T_{switch} を τ の実行時間に考慮する必要がある．また (I) においても，現在のタスクの実行直前にコンテキストの切り替えが行われたことを知る必要がある．まず，コンテキスト切り替え順序を調べるため，以下のような各タスクが実行を開始する順序を表すための変数を定義する.

- 0-1 整数変数 $seq_{p,\tau,\tau'}$: 実行系列 p 内でタスク τ の次に τ' が実行開始されるとき 1 となり，それ以外は 0 となる
- 0-1 整数変数 $init_{p,\tau}$: 実行系列 p 内でタスク τ が最初に実行を開始するタスクであるとき 1 となり，それ以外なら 0 となる
- 0-1 整数変数 $last_{p,\tau}$: 実行系列 p 内で τ が最後に実行を開始するタスクであるとき 1 となり，それ以外なら 0 となる

$seq_{p,\tau,\tau'}$ は，$t_{p,\tau} > t_{p,\tau'}$ のとき 0 とならなければならない.

$$(t_{p,\tau} > t_{p,\tau'}) \rightarrow (seq_{p,\tau,\tau'} = 0)$$

これを線形式で以下のように表すことができる.

$$t_{p,\tau'} + N_t \geqq t_{p,\tau} + N_t \times seq_{p,\tau,\tau'} \quad (1)$$

また seq を 1 にするための線形制約式，$init$，$last$ を定める線形制約式は以下で表すことができる.

$$last_{p,\tau} + \sum_{\tau'(\tau \neq \tau')} seq_{p,\tau,\tau'} = 1 \quad (2)$$

$$init_{p,\tau} + \sum_{\tau(\tau \neq \tau')} seq_{p,\tau,\tau'} = 1 \quad (3)$$

$$\sum_{\tau} init_{p,\tau} = 1 \quad (4)$$

$$\sum_{\tau} last_{p,\tau} = 1 \quad (5)$$

$$seq_{p,\tau,\tau'} + seq_{p,\tau',\tau} \leqq 1 \quad (6)$$

次に (I)，(II) それぞれを直接考慮するために以下の変数を導入する.

- 0-1 整数変数 $same_{p,\tau,\tau'}$: 実行系列 p 内でタスク τ の次に実行されるタスク τ' が，τ と同じコンテキストに割り当てられているなら 1 となり，それ以外なら 0 となる
- 0-1 整数変数 $co_{c,p,\tau}$: 実行系列 p を実行しているとき，p 内のタスク τ が実行を開始する時刻（$t_{p,\tau}$）にコンテキスト c を選択して実行するならば 1 となり（このとき $cs_{c,p,\tau} = 1$ でなければならない），それ以外なら 0 となる

まず，ある実行系列 p 内のタスク τ と τ'，およびコンテキスト c において，$co_{c,p,\tau}$ は以下のように表すことができる.

$$co_{c,p,\tau} \leqq cs_{c,p,\tau} \quad (7)$$

$$\sum_{1 \leqq c \leqq N_c} co_{c,p,\tau} = 1 \quad (8)$$

次に，ある実行系列 p 内のタスク τ と τ' において，$same_{p,\tau,\tau'}$ は $seq_{p,\tau,\tau'}$ および $co_{c,p,\tau}$ を用いて以下のように表すことができる.

$$\big((seq_{p,\tau,\tau'} = 1) \wedge \exists c(co_{c,p,\tau} = 1 \wedge co_{c,p,\tau'} = 1)\big) \rightarrow (same_{p,\tau,\tau'} = 1)$$

この制約式は線形でないため，以下のように線形制約式に変形する.

$$same_{p,\tau,\tau'} \leqq seq_{p,\tau,\tau'} \quad (9)$$

$$same_{p,\tau,\tau'} \leqq co_{c,p,\tau} \quad (10)$$

$$co_{c,p,\tau} - co_{c,p,\tau'} + (1 - same_{p,\tau,\tau'}) \geqq 0 \quad (11)$$

$$co_{c,p,\tau'} - co_{c,p,\tau} + (1 - same_{p,\tau,\tau'}) \geqq 0 \quad (12)$$

$seq_{p,\tau,\tau'}$ の定義において，実行系列 p において τ' が τ の次に実行を開始しないときには $seq_{p,\tau,\tau'}$ は必ず 0 になり，τ' が τ) の次に実行を開始するときには $seq_{p,\tau,\tau'}$ は 0 または 1 の値を取る．τ が $\tau_{prev}(\tau')$ のときに $seq_{p,\tau,\tau'}$ が 0 ならば，τ' の実行開始直前に必ずコンテキストの切り替えが行われることを意味する．後述の説明の簡便性のため，それを表す新しい変数として以下を導入する.

- 0-1 整数変数 $same'_{p,\tau}$: 実行系列 p 内のタスク τ は，p 内でその直前に実行されるタスクと同じコンテキストに割り当てられているなら 1 となり，τ の直前に必ずコンテキストの切り替えが必要ならば 0 となる

$same'$ は $same$ を用いて以下のように表される．

$$same'_{p,\tau'} = \sum_{\forall \tau \in p} same_{p,\tau,\tau'} \qquad (13)$$

次に，(II) において各タスクの実行所要時刻に正確にコンテキスト切り替え時のオーバーヘッドを考慮するために，タスクの実行中にコンテキストの切り替えが起こる回数をカウントする整数変数を導入する．

- 整数変数 $sw_{p,\tau}$: 実行系列 p 内のタスク τ が複数のコンテキストに重複して割り当てられているとき，τ の実行中に起こるコンテキスト切り替え回数を表す

$sw_{p,\tau}$ を用いることで，実行系列 p におけるタスク τ のコンテキスト切り替え回数分のオーバーヘッドを考慮したタスクの実行所要時間 $exe_{p,\tau}$ は以下の線形制約式で表せる．

$$exe_{p,\tau} = T_{exe}(\tau) + sw_{p,\tau} \times T_{switch} \qquad (14)$$

今まで実行系列 p のタスク τ が実行中の時刻範囲は $[t_{p,\tau}, t_{p,\tau} + T_{exe}(\tau)]$ としていたが，$exe_{p,\tau}$ を用いることで，コンテキスト切り替え時のオーバーヘッドを考慮した実行中の時刻範囲は $[t_{p,\tau}, t_{p,\tau} + exe_{p,\tau}]$ となる．そのため，τ は先行タスク $\tau'(=\tau_{prev}(\tau))$ の終了以降にしか実行できないという先行研究 [4] での制約は，実行系列 p において，$t_{p,\tau'} + T_{exe}(\tau') < t_{p,\tau}$ から $t_{p,\tau'} + exe_{p,\tau'} < t_{p,\tau}$ に変更する．

最後に，$sw_{p,\tau}$ を求めるために，以下の変数を導入する．

- 0-1 整数変数 $swc_{p,\tau,\tau'}$: ある実行系列 p 内のタスク τ' の実行開始時刻 ($t_{p,\tau'}$) において，まだ τ が実行中であり，かつ，τ' の実行開始直前にコンテキストの切り替えを行う必要があるなら 1 となり，それ以外なら 0 となる

この $swc_{p,\tau,\tau'}$ は，$same_{p,\tau,\tau'}$ を用いて以下のように表される．

$$\big((t_{p,\tau} < t_{p,\tau'} < t_{p,\tau'} + exe_{p,\tau}) \wedge (same'_{p,\tau'} = 0)\big) \rightarrow (swc_{p,\tau,\tau'} = 1) \qquad (15)$$

これは線形式でないため，先行研究 [4] において定義した $t_{p,\tau} < t_{p,\tau'}$ のとき 1，それ以外は 0 になる 0-1 整数変数 $tc_{p,\tau,\tau'}$ を用いて以下のように線形制約式に変形することができる．

$$t_{p,\tau'} + exe_{p,\tau} \leqq t_{p,\tau'} + N_t \times \big(same'_{p,\tau'} + swc_{p,\tau,\tau'} + (1 - tc_{p,\tau,\tau'})\big) \qquad (16)$$

実行系列 p 内のタスク τ 実行中のコンテキスト切り替え回数を表す $sw_{p,\tau}$ は，$swc_{p,\tau,\tau'}$ を用いて，以下の線形制約式で表すことができる．

$$sw_{p,\tau} = \sum_{\tau' \in p} swc_{p,\tau,\tau'} \qquad (17)$$

以上により，コンテキスト切り替え時の再構成オーバーヘッドを考慮したマルチコンテキスト型動的再構成可能プロセッサを全て線形制約式で表現することができ，ILP として解くことが可能である．

5. 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．このため，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．提案するヒューリスティックアルゴリズムでは，タスクの実行時間を利用し，以下の 5 ステップでタスクのスケジューリングを行う．

（1） 同期関係，先行関係でシステムを独立なプロセス群に分割する

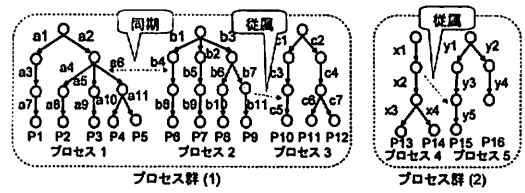

図 3 独立なプロセス群に分割

（2） 各独立なプロセス群それぞれにおいて，実行可能な全てのパスを抜き出す

（3） 各独立なプロセス群それぞれにおいて，実行可能なパスとタスクの直積からノードを作成する

（4） 全プロセス群のノードを集め，時間的に同時に動作する可能性があるかどうかでエッジを引いたグラフにする

（5） 得られたグラフを元にコンテキストに分割する

5.1 独立なプロセス群への分割

あるタスクが，同期先，および，時間制約中で他のプロセスのタスクの実行開始時刻を参照しているとき，それらのタスクのスケジューリング時刻に依存関係が発生する可能性がある．このような依存関係があるタスクを元に，プロセス間の依存関係を調べる．依存関係がないプロセス同士は，独立してスケジューリングすることができる．換言すると，依存関係のあるプロセス同士では，それらのプロセス内のパスの組合せ全てを考慮したスケジューリングが必要であるが，依存関係がないプロセス間では，パスの組合せを取る必要がなく，独立してスケジューリングすることが可能になる．1 プロセス内のパス数を p，プロセスの数を n とした場合，考慮すべき実行系列数は $O(p^n)$ となる．このとき，独立なプロセス群数が m であり，$m = kn$（k は定数，$0 < k < 1$）とした場合，考慮すべき実行系列数は $O(p^{1/k})$ まで削減することが可能である．

独立なプロセス群へ分割する例を図 3 に示す．$b7$ と $c5$，$x1$ と $y5$ にはそれぞれ時間制約中での開始時刻の参照があり，$a6$ と $b4$ には同期関係があるため，プロセス 1，2 と 3，プロセス 4 と 5 にはそれぞれ依存関係がある．従って，プロセス群 (1) とプロセス群 (2) のように，独立なプロセス群として分割できる．

5.2 実行可能な実行系列の導出

それぞれの独立なプロセス群の各プロセスから，タスクのパスを抜き出し，同期関係，実行開始時刻の参照関係から時間制約を満たして実行可能なパスの組合せ（実行系列）を導出する．これは，システムをコンテキスト単位に分割せずにコンテキスト 1 枚の動的再構成可能プロセッサに実装したときに実行可能なパスの組み合わせ（実行系列）と等価である．図 3 のプロセス群 (1) の場合，プロセス 1 から 3 の各プロセスのパスの組合せに同期関係，参照関係を考慮したものが，実行可能な実行系列となる．この例では，タスク $a6$ と $b4$ に同期関係があるため，パス $P4$，$P5$ は $P6$ と必ず並行に動作する．また，タスク $b7$ とタスク $c5$ に参照関係があるため，パス $P9$ は $P10$ と必ず並行に動作する．従って，このプロセス群の実行可能な実行系列は，次の 19 通りとなる．$(P1, P7, P11)$, $(P1, P7, P12)$ $(P1, P8, P11)$, $(P1, P8, P12)$, $(P1, P9, P10)$, $(P2, P7, P11)$, $(P2, P7, P12)$, $(P2, P8, P11)$, $(P2, P8, P12)$, $(P2, P9, P10)$, $(P3, P7, P11)$, $(P3, P7, P12)$, $(P3, P8, P11)$, $(P3, P8, P12)$, $(P3, P9, P10)$, $(P4, P6, P11)$, $(P4, P6, P12)$ $(P5, P6, P11)$, $(P5, P6, P12)$.

5.3 コンテキストへのタスク割り当てグラフの作成

5.3.1 ノード

あるプロセスのタスクとまた別のプロセスのタスクの実行時間が重複する場合，それらは同じコンテキストに割り当て，並行に動作させなければならない．しかし，実行系列毎にタスクの実行開始時刻が異なり，そのタスクが並行に動作する他のプ

ロセスのタスク群も実行系列毎に異なる．そのため，5.2 節で導出した実行系列とタスクの組合せから，コンテキストに割り当てるタスク集合を表すノードを作成する．

5.3.2 エッジ

各ノード d において，そのノードに対応するタスクが時間制約を満たして最も早く実行を開始できる時刻 $t_{est}(d)$，最も遅くに実行を開始できる時刻 $t_{lst}(d)$，そして，そのノードが実行に要する時間 $T_{exe}(d)$ とする．図 4 に示すように，もし $t_{lst}(d) < t_{est}(d) + T_{exe}(d)$ ならば，時刻 $[t_{lst}(d), t_{est}(d) + T_{exe}(d)]$ においてノード d に対応するタスクは実行中である．それ以外の時刻 $[t_{est}(d), t_{lst}(d) + T_{exe}(d)]$ は，そのタスクが実行中になる可能性がある時刻である．実行中の時刻範囲が重なるノード同士は同じコンテキストに割り当てられなければならない．また，実行可能時刻範囲が全く重ならないノード同士が同じコンテキストに割り当てられると，同時に動作することはないためプロセッサの面積効率が悪くなる．これを利用して，全プロセス群の全実行系列の各タスクに関するノードを集め，各ノード間について，実行可能時刻範囲を比較し，以下の 3 つの場合に応じてエッジを設定しグラフを作成する．

- 実行中の時刻範囲が一部分でも重なる：二重辺
- 実行中になる可能性のある時刻範囲が一部分でも重なる：一重辺
- 実行中になる可能性のある時刻範囲が全く重ならない：エッジなし

ここで，コンテキスト切り替え時のオーバーヘッド T_{switch} を考慮するため，各ノードの実行に要する時刻 $T_{exe}(d)$ は，対応するタスク τ の実行所要時刻 $T_{exe}(\tau)$ に T_{switch} を加えたものとする．

図 5 は，ノード 1，2，3，4 の実行中になる時刻範囲と，その時刻範囲に応じてエッジを追加した例である．ノード 1 とノード 2 は，必ず実行が行われる時刻が重なるため，ノード間に二重辺を引く．ノード 3 は，ノード 1，ノード 2，ノード 4 とそれ以外の実行時刻範囲が重なるためそれぞれのノードとの間に一重辺を引く．それ以外はノード間で実行時刻範囲が重ならないため，エッジを追加しない．

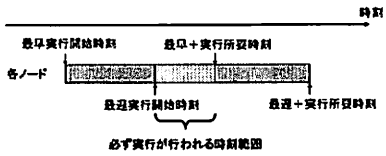

図 4 ノードのスケジューリング可能時刻範囲

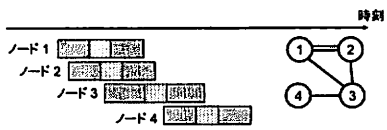

図 5 実行可能時刻範囲に応じたエッジの追加例

5.3.3 コンテキストに分割

アルゴリズムの基本方針は，クリークになっている二重辺で結ばれたノード群は必ず同じコンテキストに載せるようにすることである．なお，5.3.2 節で作成したグラフは，以下のような性質を持つ．

- 一度引かれた二重辺はスケジューリングによって変化しない
- 一重辺は二重辺，もしくは無辺になる可能性を持つ
- 辺のないノード間には スケジューリング後に辺が追加されることはない

この性質に基づき，連結なグラフのサイズを等分するように，一重辺のカットセットを求め，その一重辺を削除し，グラフを分割する．このとき，カットセットの端点となるノード群の実行可能時刻範囲の共通部分は必ず発見され，その共通部分の時刻範囲内の一点でコンテキストの切り替えが行われる．ここでは簡単のため，その時刻範囲の中央でコンテキストの切り替えが行われるとし，それらのノードの新しい実行可能時刻範囲を求め，他の一重辺が二重辺になるかどうかを再調査する．

また，図 6 のように，ノードを 2 つに分割することでグラフを分割することも可能である．このとき，このノードに対応するタスクは複数のコンテキストへ重複して割り当てられることになる．

グラフが全て二重辺のクリークになるまで分割し，最小単位のコンテキスト割り当てを求める．

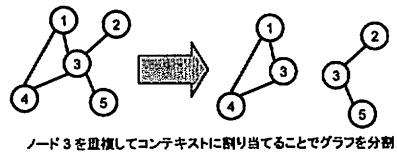

図 6 グラフの分割

最後に，分割したコンテキストの枚数が対象とする動的再構成可能プロセッサのコンテキスト枚数以下になるように，小さいコンテキスト同士をマージする．

6. 評価実験

6.1 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，ランダムな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の解を比較した．ILP を解くソルバーとして GLPK [8] を使用し，また，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 Core(TM)2 Duo 1.2GHz，メモリ 1.99GB，Windows XP Professional の計算機を用いた．

各手法の計算時間と解の精度を比較するため，タスク数を増加させたときの各手法の計算時間と最大コンテキストサイズを比較した．$N_c = 4$，$T_{switch} = 2$ とした時の計算時間の比較結果が表 1，最大コンテキストサイズの比較結果が表 2 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ことを表す．結果はそれぞれ 10 個のランダムなタスクグラフから平均を算出した．

表 1 各手法の平均計算時間の比較

	2 × 2	3 × 3	4 × 4
ILP	0.16sec	14min	>2h
ヒューリスティック	31ms	47ms	63ms

表 2 各手法の最大コンテキストサイズの比較

	2 × 2	3 × 3	4 × 4
ILP	6.5	11.9	19.7
ヒューリスティック	9.1	16.1	23.6

表 1 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加しているのに対し，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．また，表 2 から，ヒューリスティックを用いた手法は，最適解である ILP の最大コンテキストサイズの 1.3 倍程度にとどまる解を導出した．このことから，比較的小さな問題に対しては ILP を，

大きな問題に対してはヒューリスティックを使うことによって精度の良い解が得られると考えられる．

6.2 システム設計における本手法の適用

本手法を図7のような機能の書き換えを想定した次世代携帯電話に適用した．

次世代携帯電話はMP3再生，GPS受信，Wi-Fi送受信，通話，動画撮影，動画再生，YouTube再生機能を持っている．次世代携帯電話は動作指示処理モジュールを持ち，そのモジュールからの指示によってMP3再生，通話，動画撮影，動画再生，YouTube再生が行われる．動作指示処理モジュールとは独立したGPS受信，Wi-Fi送受信機能を持っており，常にGPS情報を取得し，必要に応じてWi-Fi送受信を行うことができる．YouTube再生時は，メモリ読み込み時間までにWi-Fi通信機能により必要な情報がロードできている必要があるため，YouTube再生機能とWi-Fi受信機能には従属関係がある．また，次世代携帯電話はデータメモリを持ち，動作指示処理はメモリに格納されている動作指示データを読み取って指示を送る．動作指示データは1周期前の状況判定処理によってデータに格納されているとする．各処理モジュールからデータメモリまではバスでつながれているが，競合が起こらないものとする．この次世代携帯電話の動作をモデル化したものが図8である．

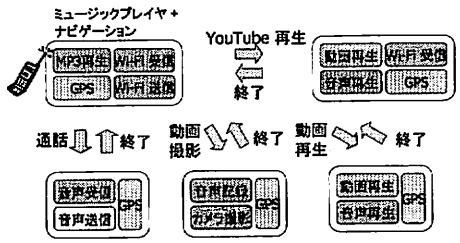

図7 次世代携帯電話動作イメージ

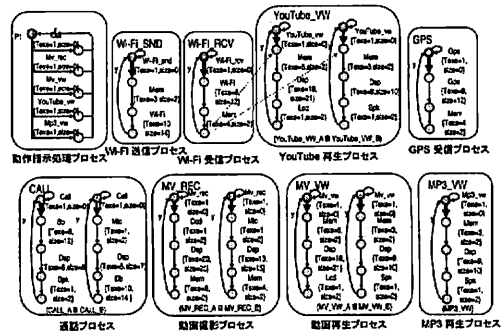

図8 次世代携帯電話のモデル

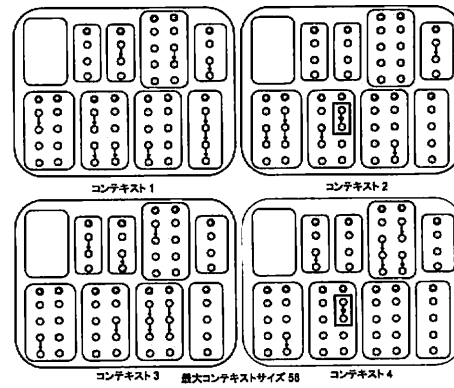

図9 次世代携帯電話の分割結果

6.2.1 アルゴリズムの実行結果

6.2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る．コンテキストにはサイズの存在するタスクの割り付けのみを考える．この例題に対してアルゴリズムを適用し図9のようなシステム分割結果を約0.2秒の計算時間で導出した．

7. おわりに

本稿では，再構成オーバーヘッドを考慮したマルチコンテキスト型動的再構成可能プロセッサに実時間システムを実装する際に，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，タスク割り当てが時間制約を満たす制約を整数線形計画 (ILP) で定式化した．次に，タスクが実行される時間を利用したヒューリスティックな方法を考案し，解の最大サイズが最適解の1.3倍程度にとどまること，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. Toi, and T. Awashima : "Stream Applications on the Dynamically Reconfigurable Processor," *Proc. of Int'l Conf. on Field Programmable Technology*(FPT2004), 2004

[2] R. Maestre, R. Hermida, F.J. Kurdahi, N. Bagherzadeh, H. Singh : "Optimal vs. Heuristic Approaches to Context Scheduling for Multi-Context Reconfigurable Architectures," *IEEE Int'l Conf. on Computer Design*(ICCD'00), 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 IFIP Int'l Conf. on Very Large Scale Integration*(VSLI-SoC 2006), 2006

[4] 木谷友哉, 中橋亮, 中田明夫, 安本慶一, 東野輝夫 : "動的リコンフィギャラブルプロセッサへの実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," 組込技術とネットワークに関するワークショップ(ETNET2007), 2007

[5] 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.

[6] NEC Electronics Corp., Dynamically Reconfigurable Processors, http://www.necel.com/ja/techhighlights/drp/

[7] IPflex, DAP/DNA-2, http://www.ipflex.com/jp/

[8] The GNU Operating System, GLPK, http://www.gnu.org/software/glpk/glpk.html