情報処理学会論文誌　コンピューティングシステム　Vol. 4　No. 4　12–23 (Oct. 2011)

再構成回数削減による動的リコンフィギャラブルプロセッサの消費電力削減手法の提案

木村優之[†1]　弘中和衛[†1]　天野英晴[†1]

本論文では動的リコンフィギャラブルデバイスのアーキテクチャを変更することなく電力を削減するためのマッピング手法を提案する．あらかじめ各再構成ユニットが同じ命令を実行するように配置を変更し，再構成回数を抑制することで，再構成ユニットで消費される電力オーバヘッドを削減する．提案アルゴリズムの効果を評価するために，提案手法を適用したコンパイラを開発し，本研究室で開発された動的リコンフィギャラブルプロセッサを用いて性能，構成情報サイズ，実行時消費電力をシミュレーションにより調査した．その結果，アプリケーションの実行時間を増加させることなく，消費電力を平均で 10%削減することができた．

Power Reduction for Dynamically Reconfigurable Processor Array with Reducing the Number of Reconfiguration

Masayuki Kimura,[†1] Kazuei Hironaka[†1]
and Hideharu Amano[†1]

A power consumption centric assignment algorithm is proposed for dynamically reconfigurable processors. By assigning the same operations to the same PE (Processing Element) as much as possible, the number of changing configuration data for dynamic reconfiguration can be reduced. The redundant power consumption for changing the configuration is also reduced as a result. The proposed algorithm is implemented in a compiler for a research dynamically reconfigurable processor MuCCRA-3, and its evaluation results showed that the power consumption was reduced by 10% in average without increasing the execution time.

1. はじめに

近年のモバイル機器の普及にともない，組み込みデバイスには性能向上，低消費電力化に加え，開発期間の短縮，柔軟性がますます要求されるようになっている．これらの要求を満足するために，専用ハードウェアに代わるオフロードエンジンとして，動的リコンフィギャラブルプロセッサ（Dynamically Reconfigurable Processor Array，DRPA）が注目されている．すでに商用化もされており，その例として SONY の VME [1)]，NEC の STP エンジン[2)]，Panasonic の D-Fabrix [3)] などがあげられる．

一般的なマルチコンテキスト型動的リコンフィギャラブルプロセッサは単純な演算器やレジスタファイルをひとまとめにした Processing Element（PE）をアレイ状に複数個並べた構成をとっている．各 PE の演算命令や PE 間結合網を決める構成情報（コンフィギュレーションデータ）を複数セット保持し（コンテキストと呼ぶ），これを実行時に高速に切り替えることでアプリケーションを実行する．

DRPA の演算単位は粗粒度であるため，専用ハードウェアや FPGA などの細粒度のデバイスに比べて C 言語などの高水準言語でのアプリケーション記述に適している．高水準言語で記述されたプログラムから構成情報を生成するためには，以下のような手順をふむ場合が多い．

(1)　プログラムの構文解析，意味解析を行い，アセンブリ命令列を生成する．
(2)　中間命令表現のデータフローグラフ，制御フローグラフを生成する．その結果に基づきスケジューリングを行い，各コンテキストで実行すべき命令を決定する．
(3)　命令のマッピングを行う．PE 不足などでマッピングができない場合は，(2) まで戻ってスケジューリングをやり直す．
(4)　データフローグラフに基づきルーティングを行う．配線資源不足でルーティングができない場合は，(2) まで戻ってスケジューリングをやり直す．
(5)　DRPA で実行可能な構成情報を生成する．

これらの処理を人間の手で行うことは困難であるため，動的リコンフィギャラブルプロセッサのアプリケーションの開発にはコンパイラが必須である．このため，様々なコンパイラが提案され，性能を向上させ，コンテキスト数や利用 PE 数を減らすためのマッピング手法などが検討されてきた[4)–7)]．

†1 慶應義塾大学大学院理工学研究科
Graduate School of Science and Technology, Keio University

動的リコンフィギャラブルプロセッサの利点の 1 つは，その消費電力が小さいことである．これは問題の解法アルゴリズムを直接 PE アレイ上で実行できるためで，この利点をさらに生かすため，様々な解析や構成上の工夫がなされている[8]．この結果，動的リコンフィギャラブルプロセッサでは，コンテキストの切替え時の構成情報を切り替えることによるデータフローの変化，すなわち演算の切替えや，演算器に入力されるオペランドの変化によって多くの電力を消費していることが明らかにされた[9]．このため，これらの消費電力を削減するアーキテクチャ上の工夫がいくつか提案されている[10]．一方，演算の切替え頻度やプログラムの性能はコンパイラの構成情報生成アルゴリズムにも依存しており，コンパイラによる構成情報の最適化は低消費電力化のために有効である．しかし，従来のコンパイラは，性能の向上や，コンテキスト数を抑えることを目標に設計されており，このような検討はほとんどなされていなかった．

そこで，本論文では，動的リコンフィギャラブルプロセッサの消費電力を抑えるための構成情報最適化手法として，Partially Fixed Configuration Mapping（PFCM）アルゴリズムを提案する．この手法ではコンテキスト切替え時に各ユニットの再構成回数を削減することにより，消費電力を削減する．

2. 関 連 研 究

動的リコンフィギャラブルプロセッサの電力効率をより向上させるために，演算器に対するオペランドアイソレーションの適用[8]，細粒度部分再構成による消費電力削減技法[10]，2 電源手法の利用[11] などが提案されている．しかし，これらの提案手法の多くは，アーキテクチャ自体の改良手法であり，適用するためには，ハードウェアの実装し直しが必要となる．

これに対してコンパイラやマッピング手法による電力節約は，ハードウェアの変更なしに効果を得ることができることから，特に FPGA 向けに様々な方法が提案されている．これらは，配線による消費電力の最適化，モジュールの再利用，高いレベルから電力を考慮して合成およびマッピングをするなど[12)–14)]，FPGA に搭載する回路方式やマッピング手法を工夫することで電力を削減している．また，2 種類の電源が選択可能な FPGA [15] や，スレッショルドレベルが異なる構成要素を持つ FPGA [16] など特殊な FPGA を利用して大きく消費電力を削減する方法も提案されている．しかし，これらはいずれも静的にマッピングされた回路の消費する電力を削減する研究であり，本研究の狙いとは異なっている．

一方，動的リコンフィギャラブルデバイス向けのコンパイラも数多く開発されている[5)–7)]．これらのコンパイラでは，FPGA 向けマッピング手法の適用[17] や，VLIW 向け並列化手法の適用[4] などを利用し，並列性を向上させ，性能を向上させることに成功している．しかし，コンパイラやマッピング手法により消費電力を削減する研究は我々の知る限りこれまでに行われていない．

3. MuCCRA-3 アーキテクチャ

本研究では，評価対象アーキテクチャとして研究用動的リコンフィギャラブルプロセッサ MuCCRA-3 [9] を用いた．MuCCRA-3 は**図 1** に示す小規模な PE アレイと単純な PE から構成される動的リコンフィギャラブルプロセッサである．

3.1 PE アレイ

MuCCRA-3 の PE は，演算を行う ALU，ALU のオペランドを選択する ALU_DATA_SEL，レジスタファイル RF，PE 間結合網を構成する SE の 4 種類の再構成ユニットを含んでいる．PE の構造と，SE の構造をそれぞれ，**図 2**，**図 3** に示す．

MuCCRA-3 は，4 × 4 の PE による 2 次元アレイ構造を持つ．PE アレイの上端と下端には 4 つずつ，計 8 つの MEM ユニットを持つ．MEM ユニットは 128 ワードのデータメモリを 2 バンク含んでおり，ダブルバッファリング方式を採用することによりデータ転送時間を隠蔽している．

PE 間を接続する結合網は，Switching Element（SE）を利用したアイランドスタイルに加えて，隣接している PE との直結網の 2 種類の結合網が存在する．図 1 の太線が直結網に

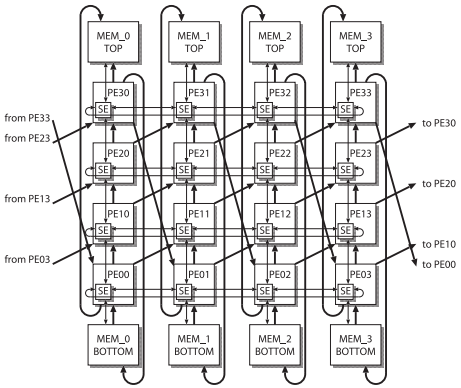

図 1　MuCCRA-3 の PE アレイ構成
Fig. 1　PE array of MuCCRA-3.

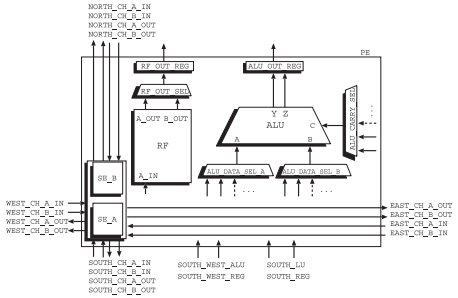

図 2　MuCCRA-3 の PE 構成
Fig. 2　PE structure of MuCCRA-3.

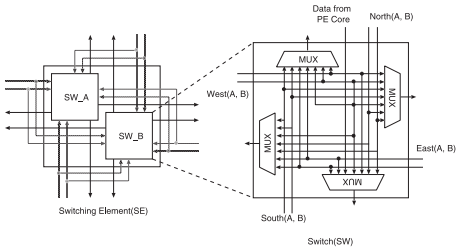

図 3　MuCCRA-3 の SE 構成
Fig. 3　SE structure of MuCCRA-3.

よる接続を，細線が SE による接続網を示している．SE による接続網は A，B の 2 チャネルが利用可能で，SE により，A から B，または B から A の乗り換えが可能となっている．

3.2　再構成制御機構

MuCCRA-3 はコンテキストと呼ばれる複数の構成情報のセットを切り替えて再構成を行うマルチコンテキスト方式を採用した DRPA である．MuCCRA-3 の再構成制御機構は，コンテキストの転送を制御する Task Configuration Controller（TCC）と，コンテキストの切替えを制御する Conetxt Switch Controller（CSC）の 2 つから構成されている．

MuCCRA-3 では，1 つのアプリケーションを複数のタスクという単位に分割して制御を行う．1 つのタスクは，最大で 32 個のコンテキストの集まりである．各タスクには，Task Flow Table（TFT）と呼ばれる，ヘッダ情報が付加されており，TCC はその情報を参照して，外部のメモリからコンテキストを読み込み，各 PE などに転送を行う．タスク実行中のコンテキストの切替えは，CSC が分岐などを判断して生成するポインタによって行う．

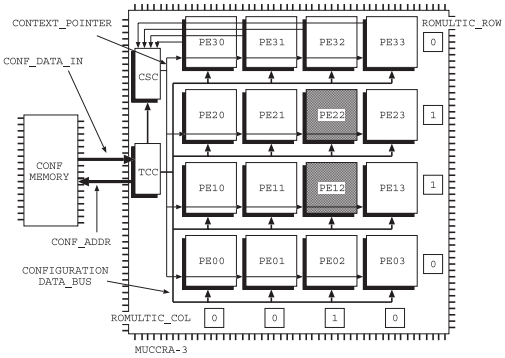

図 4　MuCCRA-3 の制御機構
Fig. 4　Reconfiguration control mechanism of MuCCRA-3.

各 PE，MEM は自身が持つコンテキストメモリから構成情報を読み出して再構成する．したがって実行に先だち，コンテキストメモリへ構成情報を転送しておく必要がある．この転送は Task Configuration Controller（TCC）が行う．**図 4** に MuCCRA-3 の制御機構を示す．まず，TCC は全構成情報を保持しているチップ外メモリに読み出しアドレス（CONF_ADDR）を与え，チップ外メモリは構成情報を返す．TCC と PE，MEM とは共有バスである CONFIGURATION_DATA_BUS で接続されており，TCC は，構成情報とともにユニットの種類も一緒に送り，PE 側は受け取るべきデータであるか判断をしてコンテキストメモリに書き込む．また，MuCCRA-1，2 同様，転送に RoMultiC[18] と呼ばれるマルチキャスト転送を行う．

RoMultiC は，行と列のマルチキャストビット（ROMULTIC_ROW，ROMULTIC_COL）を用いる転送手法である．コンフィギュレーションデータは，行と列の該当ビットがともに 1 である場合（図 3 の例では PE12 と PE22），コンテキストメモリに書き込まれる．これにより複数のモジュールへ同一のコンフィギュレーションデータを転送する場合に，転送時間を短縮することができる．

3.3　電力の予備評価

まず，動的リコンフィギャラブルプロセッサのデータパスの切替えに要する消費電力を調査するために，以下の 2 種類のアプリケーションの消費電力を MuCCRA-3 のポストレイアウトシミュレーションにより解析した．**図 5** に各アプリケーションの演算の配置について示す．

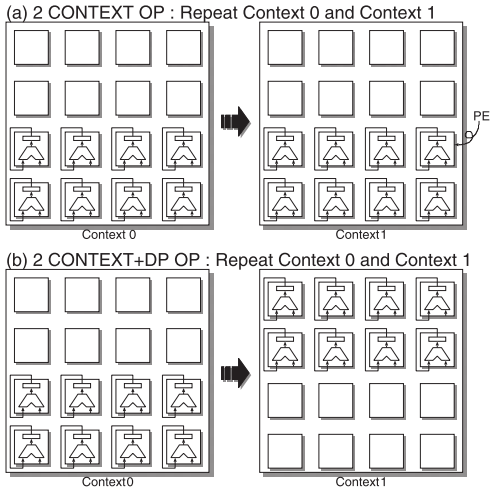

図 5　データパス切替え時のオーバヘッドを調査するための構成情報
Fig. 5　An example mapping for the pre-evaluation of datapath reconfiguration overhead.

2 CONTEXT OP　2 つのコンテキストを切り替えながら実行する．どちらのコンテキストもアレイの下半分を利用して演算を行い，レジスタへ代入する．各 PE の構成情報はコンテキストを切り替えても変化しない．

2 CONTEXT+DP OP　PE の演算内容は “2 CONTEXT OP” と等しいが，コンテキストが切り替わると動作する PE がアレイの上半分と切り替わる．アプリケーションの演算は変化しないが，各 PE の構成情報はコンテキストが切り替わるごとに変化する．

加算，乗算，右シフト，左シフト，バレルシフトの各演算について上記のアプリケーションをそれぞれ実行し，消費電力を評価したときの結果を**図 6** に示す．すべての演算で 2 CONTEXT+DP OP は 2 CONTEXT OP よりも消費電力が大きく，特に乗算命令を実行した結果では 30%の電力オーバヘッドが生じていることが分かる．このオーバヘッドはコンテキストメモリから構成情報を読み出し，演算内容や演算データを切り替えて多くの配線がスイッチされるために生じるものである．

このシミュレーション結果より，再構成ユニットの構成情報はなるべく変化させず，ユニットの構成情報の切替え回数を削減することが消費電力を抑えるために重要であることが分かる．

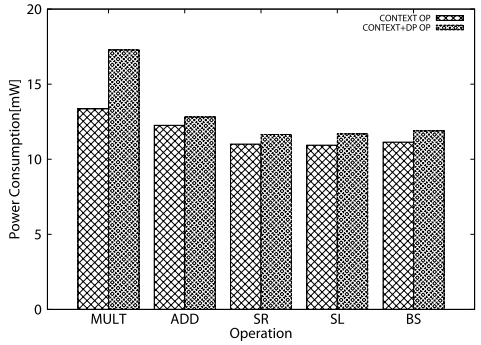

図 6　2 パターンのコンテキストでの消費電力
Fig. 6　Consuming power in two different mappings.

4. 提案手法：PFCM アルゴリズム

3.3 節の評価結果より，再構成ユニットが構成を切り替えることによる電力オーバヘッドを削減するために，なるべく構成情報を切り替えずに固定することが有効であることが分かった．

コンテキストの切替えが生じても再構成ユニットの構成情報を維持するためには，コンパイラによりアプリケーションのスケジュールを調整し，あるコンテキストで利用しない再構成ユニットが存在する場合，1 つ前のコンテキストの構成情報を設定することで，見かけ上再構成を行うことなく，処理を実行することが可能である．このスケジューリングを実現するためのアルゴリズムを Partially Fixed Configuration Mapping（PFCM）アルゴリズムと呼ぶ．PFCM はアプリケーション性能への影響を最小限に抑え，演算の再配置と構成情報の伝搬を行うことでアプリケーションの消費電力を削減することができるアルゴリズムである．

PFCM アルゴリズムは，大きく 2 つの段階に分けられる．

- 演算の再配置
- 構成情報の伝搬

まず，演算の再配置では，次段階の構成情報の伝搬処理が効果的に適用できるようにするため，演算命令の再配置を行い，各 PE が頻繁に命令の種類を変更することを防ぐ．

次に，構成情報の伝搬処理を行い，再構成ユニットの構成情報切替え回数を削減する．

各段階について，以下で詳細なアルゴリズムを説明する．

4.1　演算の再配置

演算の再配置処理では，あらかじめ各再構成ユニットが同じ構成情報を利用することができるよう，演算の配置を変更する．

図 7 に PFCM の演算の再配置処理アルゴリズムを示す．

PFCM を適用するためには，まずスケジューリング，配置配線が行われた構成情報を用意し，その構成情報に対して最適化を行う．

ここでは，各再構成ユニットに以下の 3 種類の状態を用意する．

- 命令が何も配置されていない（NONE）
- 意味のある命令が配置されている（VALID）
- 意味のない命令が配置されている（INVALID）

まず，再配置のために，アプリケーション全体で利用される演算の種類，数，実行される場所を調査する．

利用される回数の多い演算命令の方が消費電力に与える影響が大きいと考えられるため，アプリケーション中で利用される回数の最も多い命令種に属する命令から順に再配置処理を行う．図 7 では，numOfInst という配列に演算回数を格納する．配列の添字は命令の種類である．解析結果に基づき，演算回数の最も多い命令種に属する命令から順に再配置を実行する．

図 7 の *kind* は，再構成ユニットで実行される命令の種類を表す．たとえば，加算命令 add，addi はどちらも再構成ユニット上では同じ加算を実行するため，同じ *kind* が割り当てられる．

extractInstructionList() を用いて *kind* に分類されている命令を取り出す．そのときの命令列を *inst_list* に格納されているとする．

命令列中の各命令を *inst* とし，各命令について再配置を行う．また命令 *inst* の実行されるコンテキスト番号を s とする．

まず，*findSameTypePE* 関数は同一コンテキスト上で命令の種類が *kind* である意味のない（状態が INVALID である）命令が配置されている最も近い再構成ユニットを探索する．**図 8** に，*findSameTypePE* 関数の動作を示す．命令種類が等しい意味のない命令がアレイ上に配置されている場合，その場所に *inst* の配置を移動する（命令の再配置）．

同一種類の命令の場所への再配置が失敗した場合，再配置可能な利用されていないユニットを探す必要がある．これがプログラム中の *findEmptyPE*() である．

findEmptyPE により利用されていない再構成ユニットが見つかった場合，その再構成ユニットのすべてのコンテキストに同一種類の命令を配置する．ただし，配置した命令はア

```
for each instruction inst
  numOfInst[ inst.kind ]++
sort( numOfInst ) in descend order
for each numOfInst
  kind = index of selected element of numOfInst
  inst_list = extractInstructionList(kind)
  for each inst_list inst
    s = scheduled context of inst
    (x0, y0) = geometry of inst
    (x, y) = findSameTypePE(x0, y0, kind)
    if failed to findSameTypePE &&
        status of allocTable_{x0,y0,s} is NONE
      call invalidate(kind, x0, y0)
      call validate(kind, x0, y0, s)
    else if success to findSameTypePE
      call validate(kind, x, y, s)
    else
      (x, y) = findEmptyPE(x0, y0, kind)
      if success to findEmptyPE
        call invalidate(kind, x, y)
        call validate(kind, x, y, s)
      else
        call validate(kind, x0, y0, s)
      endif
    endif
  endfor
endfor
function validate(kind, x, y, s)
  set allocTable_{x,y,s} as kind
  set status of allocTable_{x,y,s} as VALID
endfunction
function invalidate(kind, x, y)
  for each context c
    set allocTable_{x,y,c} as kind
    set status of allocTable_{x,y,c} as INVALID
  endfor
endfunction
```

図 7　構成情報の再配置アルゴリズム
Fig. 7　The algorithm of instruction reallocating.

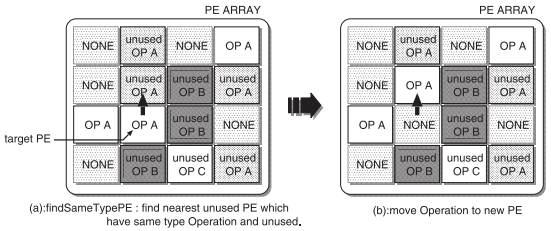

図 8　命令の再配置処理（$findSameTypePE()$）
Fig. 8　Instruction reallocation ($findSameTypePE()$).

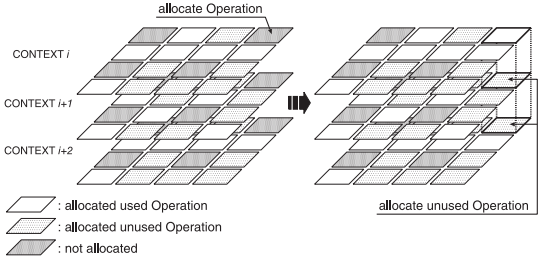

図 9　命令の再配置と意味のない命令の配置
Fig. 9　Padding INVALID instructions.

プリケーション実行においてはすべて意味のない命令である．**図 9** に，命令の再配置動作の概要を示す．CONTEXTi に命令が配置され，CONTEXT$i+1$，CONTEXT$i+2$ にも同一の命令が配置される．

CONTEXT$i+1$，CONTEXT$i+2$ に配置された意味のない命令は，それぞれのコンテキストでの $findSameTypePE()$ で利用される．

一方，$findEmptyPE$ 関数は利用されていない再構成ユニットを探索する．**図 10** に，$findEmptyPE$ 関数の動作を示す．利用されていないユニットが見つかった場合，そのユニットに命令を配置する．

利用されていないユニットが見つからなかった場合，その命令が初期配置されている再構成ユニットに，強制的にその命令を配置する．強制的な配置により，その再構成ユニットでは実行中に構成情報の切替えが生じてしまう．これは他に利用できる PE がないためのやむをえない処置である．

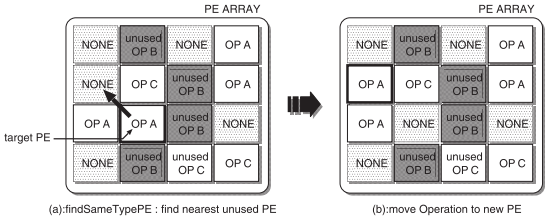

図 10　何も配置されていないユニットへの命令の再配置（$findEmptyPE()$）
Fig. 10　Reallocate instruction to unused PEs ($findEmptyPE()$).

```
pred_s = 0;
s = 1;
while all region of x and y
  while all region of s
    if status of allocTable_{x,y,s} is not VALID &&
          inst of PE[x, y, s] != inst of PE[x, y, pred_s]
      propagate_conf( inst of PE[x, y, s],
                      inst of PE[x, y, pred_s]);
    endif
    pred_s = s;
    s = next context of s
  endwhile
endwhile
```

図 11　構成情報の伝搬アルゴリズム
Fig. 11　The algorithm of configuration data spreading.

4.2　構成情報の伝搬

構成情報の伝搬処理では，コンテキスト i で各再構成ユニットが利用されているか調査する．ユニットが利用されていない場合，コンテキスト i の構成情報を 1 つ前のコンテキストの構成情報で置き換える．

図 11 に構成情報の伝搬アルゴリズムを示す．

プログラムの実行順に構成情報の伝搬処理を行っていく．再構成ユニットの状態が INVALID もしくは NONE である場合，1 つ前のコンテキストで実行された構成情報をそのまま引き継ぐ（アルゴリズム中の $propagate\_conf()$）操作を，すべての構成情報に対して適用する．

表 1　再構成ユニットにおける PFCM の適用方針
Table 1　The applying policy of PFCM for reconfiguration units.

再構成ユニット	PFCM の適用方針
ALU	オペコードなど，すべての構成情報を 1 つ前のコンテキストの構成情報で置き換える．
ALU_DATA_SEL	オペランドなど，すべての構成情報を 1 つ前のコンテキストの構成情報で置き換える．
RF	読み込み・書き込みアドレスを 1 つ前のコンテキストの構成情報で置き換える．書き込み信号は無効にする．
SW_A，SW_B	構成情報を引き継がずに，0 を出力する．
MEM	読み込み・書き込みアドレスを 1 つ前のコンテキストの構成情報で置き換える．書き込み信号は無効にする．
CONTROL	構成情報を引き継がない．

このとき，*propagate_conf*() では，一律に構成情報を次のコンテキストに複写するのではなく，再構成ユニットごとに構成情報の最適な置き換え方針を決める．

表 1 に再構成ユニットごとの置き換えの方法についてまとめる．

ALU では，命令の種類やキャリーインの選択など，すべての構成情報を引き継がせる．ALU_DATA_SEL ユニットでは，同じくすべての構成情報を引き継ぐ．

RF ユニットも同様に大部分の構成情報を引き継ぐが，RF への書き込み信号は伝搬させずに無効化する．これは，1 つ前のコンテキストでレジスタ書き込みが行われていた場合，その結果をそのまま伝搬させると後続のすべてのコンテキストで意図しない書き込みが生じることを防ぐためである．

SW_A，SW_B ユニットに対しては，構成情報の伝搬は行わず，無意味なデータ転送命令はすべて無効にし，スイッチの出力を停止する構成情報に置き換える．これは，スイッチングエレメントはデータ転送用のユニットであるため，構成情報を引き継がせた場合無意味なデータが結合網上を流れ，消費電力が増大してしまうためである．

同様に MEM ユニットについても，DMEM への書き込み信号のみ伝搬させずに無効とする．

CONTROL ユニットは，デバイスの制御を司るユニットであり，デバイス内に 1 つしか存在しないことから，構成情報は引き継がない．

演算命令の再配置では，コンテキストをまたいだ演算命令の移動は行わない．実行される演算命令のコンテキストを変えてしまった場合，演算命令の再スケジュールの必要が生じてしまい，性能が低下する恐れがある．PFCM ではコンテキスト内でのみ演算の移動処理を実行するため，アプリケーションのコンテキスト数が変化することはない．したがって，実行サイクル数は変化せず，性能に影響を与えることなく再構成回数を最小化することができる．

5. 評　　価

5.1 評価環境

提案手法の評価のため，MuCCRA-3 を富士通 65 nm プロセスライブラリを用いて論理合成を行った．ゲートレベルシミュレーションの結果を用いて電力評価を行った．すべての手法で共通の動作周波数制約として 40 MHz を与え，これを満たすことを確認した．

評価対象のアプリケーションには α ブレンダ，グレイスケールフィルタ，セピアフィルタ，SSD（差分 2 乗和），2 次元離散コサイン変換（2D-DCT），2 次元逆離散コサイン変換（2D-IDCT）を利用した．

提案手法を評価するために，MuCCRA-3 用のコンパイラを実装した．本コンパイラでは，C 言語で記述されたプログラムを入力すると，フロントエンド処理，スケジューリング，配置，提案アルゴリズムである演算ノードの再配置処理を自動的に行う．本コンパイラでは，配線処理は行わず，後述する MuCCRA-3 用アセンブラを用いて配線を行い，提案アルゴリズムである構成情報の伝搬処理を自動的に行い，構成情報を出力する．

また，本研究では，構成情報出力アルゴリズムとして以下の 3 種類を用意し，それぞれ比較を行った．

1. **Greedy アルゴリズム**　演算命令を PE アレイの下側から順に配置する．配線が不可能になるとコンテキストを切り替える．本研究室で開発された Black-Diamond コンパイラ[6] が本アルゴリズムを利用している．
2. **Quadratic & Mincut Placement（QPlace & Mincut）**　VLSI の配置アルゴリズムである Quadratic Placement[19] と Mincut Placement[20] を交互に適用する．本研究で開発したコンパイラが利用する配置アルゴリズムである．
3. **QPlace & Mincut & PFCM**　上記の配置結果に対して本研究の提案手法を適用する．

1，2 は，提案手法を適用せず，スケジュール・配置配線のみを実行して構成情報を生成するものである．そして 2 と 3 の評価結果を比較し，提案手法の効果を検証する．

5.2 再構成頻度

提案手法の適用により再構成ユニットの再構成回数の変化について調査するため，各再構

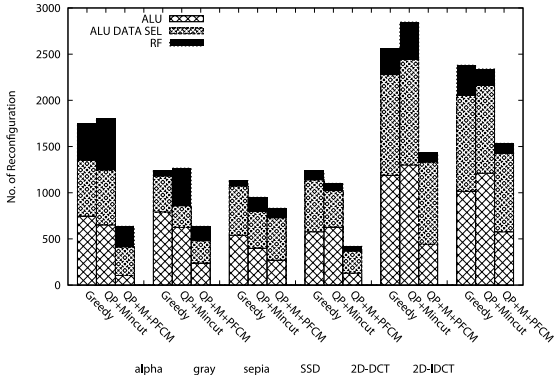

図 12　再構成ユニットごとの構成情報の変更回数
Fig. 12　The number of reconfiguration by every reconfigurable units.

成ユニットの再構成回数のシミュレーションを行った．

図 12 に，ALU，ALU_DATA_SEL，RF ユニットの PFCM 適用時と非適用時の再構成回数を示す．なお，今回構成情報の伝搬処理を行うのは上記の 3 つのユニットのみであるため，SW_A，SW_B については再構成回数のシミュレーションを行っていない．

すべてのアプリケーションにおいて，提案手法を適用した場合にすべての再構成ユニットで再構成回数が削減された．特に ALU ユニットの再構成回数の削減率が高く，α-Blender では Greedy アルゴリズムでの再構成回数に比べて 86%程度再構成回数が削減できている．

一方で，sepia フィルタではすべてのユニットで再構成回数の削減率が小さく，Greedy アルゴリズムに比べて 26%程度の削減率にとどまっている．これは，sepia フィルタは各コンテキストの再構成ユニットの利用率が低く，半分以上の PE を利用せずにアプリケーションが実行されているため，利用されていない再構成ユニットでは構成情報の変更が行われず，大部分のユニットが再構成を行っていないためである．

図 13 (a) に，2D-DCT における PFCM を適用しない際の演算ユニットの構成情報出力結果，図 13 (b) に，PFCM を適用した際の演算ユニットの構成情報出力結果を示す．

2D-DCT は演算数が多いため，構成情報の切替え回数も多く，PFCM を適用しない場合，すべてのコンテキストで構成情報の切替えが生じている．一方，PFCM を適用した場合では，非適用時に比べて再構成回数が減少していることが分かる．同アプリケーションでは，加算 26 命令，乗算 16 命令，シフト 10 命令が利用されており，1 度も再構成を行わないように PE アレイ上に配置することは不可能である．しかし同一種類の命令を同じ PE 上で実行するように演算を移動することにより，PFCM 非適用時に比べて再構成回数を削減することができている．

結論として，提案手法はある程度規模の大きなアプリケーションで，PE アレイの利用率が高い場合に効果があるといえる．

5.3　構成情報データサイズ

構成情報データサイズはアプリケーション実行前の構成情報転送時間に大きく影響する．本提案手法を適用したことによる，構成情報データサイズへの影響について検証を行った．

図 14 に，各アプリケーションにおいて Greedy アルゴリズムを用いた場合，QPlace & Mincut のみを用いた場合，QPlace & Mincut & PFCM を適用した場合について構成情報データサイズを示す．

QPlace & Mincut のみを適用した場合，Greedy アルゴリズムを適用した場合に比べてデータサイズはわずかな増加にとどまっており，2D-DCT においては減少する場合も見られた．これは MuCCRA-3 に採用されているコンフィギュレーションデータ転送アルゴリズムである RoMultiC の利用効率に強く依存しており，構成情報の PE アレイの配置方法にも強く依存する．

一方，PFCM を適用した場合，データサイズは 2 倍近く増加した．この原因としては提案手法の構成情報の伝搬処理があげられる．PFCM を適用しない場合，あるコンテキストでの構成情報は，RoMultiC のアルゴリズムにより PE の利用していない部分を活用して広く囲むことができ，その結果構成情報を小さくすることができていた．ところが，PFCM を適用することで，各コンテキストにおいて多くの PE を細かく制御する必要があり，その結果として RoMultiC による構成情報の圧縮効果が薄れる結果となり，全体として構成情報データサイズの増加を招いたと考えられる．

しかし，動的リコンフィギャラブルデバイスは 1 度構成情報を転送してしまうと，アプリケーションを切り替えるまでコンフィギュレーションデータを転送する必要がない．したがって，アプリケーションにつき処理すべきデータが多いほど，構成情報データの転送時間は隠蔽されることとなる．動的リコンフィギャラブルデバイスはメディアストリーム処理向けアクセラレータであるため，大量のデータを処理するアプリケーションを実行することが多い．したがって，小さなデータサイズのアプリケーションを頻繁に切り替えて実行しない限り，性能への影響は少ないと考えられる．

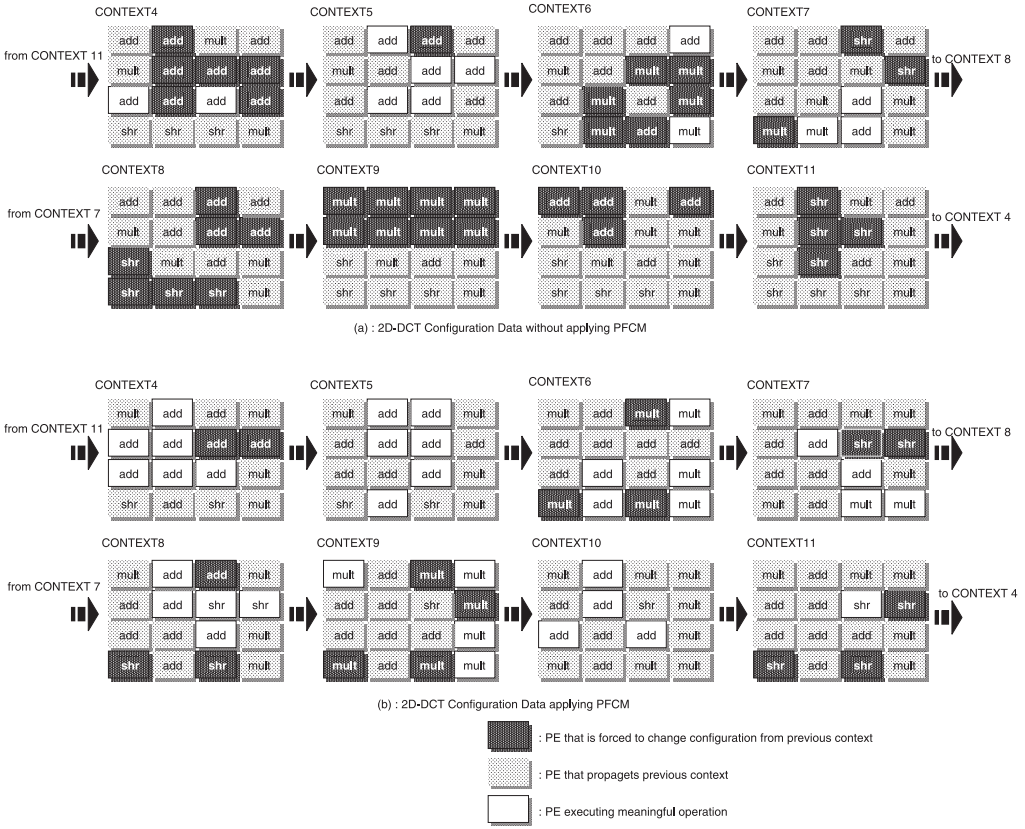

図 13　2D-DCT の PFCM 非適用時 (a) および適用時 (b) の構成情報の遷移
Fig. 13　An example of PFCM applying in 2D-DCT application.

5.4　アプリケーションの実行サイクル数

提案手法を適用したことによるアプリケーションの実行速度を解析するために，評価アプリケーションの実行サイクルのシミュレーションを行った．

表 2 に，各アプリケーションの実行サイクルを示す．

表より，QPlace & Mincut に対して PFCM を適用しても，実行サイクルに変化は生じていないことが分かる．つまり，PFCM を適用してもアプリケーションの実行速度への影響は少ない．

Greedy アルゴリズムと比べて α-Blender のみ実行サイクル数がわずかに変化しているが，これは繰返しループより外側の初期化の行い方が異なっているためである．

アプリケーションの実行速度が低下しない理由は，PFCM は構成情報の再配置を行うだけであり，再配置範囲は同一コンテキスト内のユニットのみを対象とし，コンテキストをまたいだ構成情報の再配置を行わないためである．構成情報を伝搬処理においても，事前にス

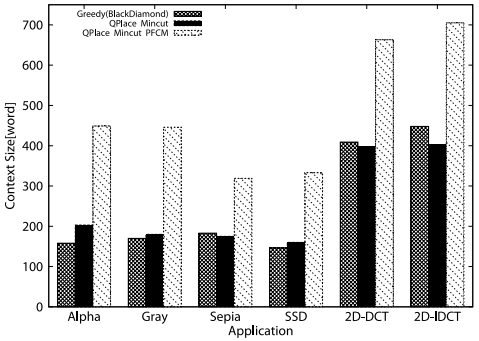

図 14　構成情報のサイズ
Fig. 14　Configuration data size of the applications.

表 2　アプリケーションの実行サイクル
Table 2　Execution cycles of the applications.

	alpha	gray	sepia	ssd	DCT	IDCT
Greedy	83	82	67	67	127	113
QPlace & Mincut	84	82	67	67	127	113
QPlace & Mincut & PFCM	84	82	67	67	127	113

ケジューリングされた構成情報に対して伝搬処理を行うだけであるため，コンテキスト数は増えない．

5.5　実行時消費電力

提案手法を適用した際のアプリケーション実行時の消費電力について評価を行った．**図 15** に，各アプリケーションにおいて Greedy アルゴリズムを用いた場合，QPlace & Mincut のみを用いた場合，QPlace & Mincut & PFCM を適用した場合について実行時の消費電力を示す．図 15 の Internal Power は，トランジスタ ON/OFF 時のセル内で消費される貫通電力である．また，Switching Power は，トランジスタに接続されている配線がトグルすることによる消費電力である．Leak Power はトランジスタのリーク電力である．ハードウェアを変更していないため，リーク電力は 3 つの手法で同一である．

Greedy アルゴリズムでは，演算に利用する PE をアレイの下側に集中させる傾向があるため，各 PE の再構成回数が多くなっており，消費電力の増加につながっているものと思われる．

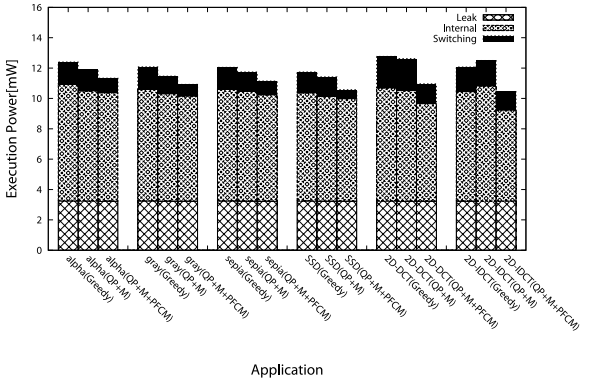

図 15　実行時消費電力
Fig. 15　Consuming power of the applications.

QPlace & Mincut を適用した場合，つまり配置配線アルゴリズムを変更した場合でも，消費電力は平均で 5%程度削減されている．これは 3.3 節の MuCCRA-3 の予備評価でも示したとおり，演算命令の配置の結果が消費電力に強く影響しているということである．

しかし，2D-DCT や 2D-IDCT のような比較的複雑なアプリケーションでは，配置配線アルゴリズムを変更しても消費電力は大幅に削減されることはない．これは PE アレイの利用率が高い場合，配置アルゴリズムごとの差が小さくなるためである．

一方 PFCM を適用した場合，平均で 10%程度の消費電力が削減できた．QPlace & Mincut ではほとんど消費電力が削減されなかったアプリケーションにおいても，PFCM を適用した場合には大幅な消費電力の削減を行うことができた．

2D-DCT のアプリケーションに注目すると，図 12 では，ALU の再構成回数が半分以上に削減されており，図 15 での 2D-DCT の消費電力が大きく削減されていることが分かる．

したがって，再構成回数を削減する本提案アルゴリズムは消費電力削減に有効であるということが分かる．

6. 結　　論

本論文では，動的リコンフィギャラブルデバイスにおける消費電力を削減するマッピング・スケジューリングアルゴリズム Partially Fixed Configuration Mapping を提案し，その効果について検証を行った．提案アルゴリズムを自動的に適用することのできる動的リコ

ンフィギャラブルデバイス向けコンパイラを新たに開発し，同提案手法を適用しその効果を検証した結果，アプリケーションの性能を低下させることなく，消費電力を平均で 10%程度削減することができた．

謝辞　本研究の一部は，科学技術振興機構「JST」の戦略的創造研究推進事業「CREST」における研究領域「情報システムの超低消費電力化を目指した技術革新と統合化技術」の研究課題「革新的電源制御による次世代超低電力高性能システム LSI の研究」による．

本研究におけるチップ開発は東京大学大規模集積システム設計教育研究センターを通し，株式会社半導体理工学研究センター，株式会社イー・シャトル，富士通株式会社，シノプシス株式会社，日本ケイデンス株式会社，メンター株式会社の協力で行われたものである．

参 考 文 献

1) Kurose, Y., Okabe, M., Seno, K., Ozawa, H., Wada, T., Taniguchi, K., Hokazono, H., Hirano, T., Kumata, I., Hanaki, H., Hasegawa, K., Horiike, S., Arima, S., Ono, K., Hiroi, T. and Takashima, S.: A 90 nm embedded DRAM single chip LSI with a 3D graphics, H.264 codec engine, and a reconfigurable processor, *Hot Chips 16* (2004).
2) Motomura, M.: C-based Programmable-HW Core “STP Engine” Current Status and Future, IEICE Technical Report, RECONF2008-48 (invited talk) (2008).
3) Panasonic: D-Fabrix, available from ⟨www.panasoniceurope.com⟩.
4) Mei, B., Vernalde, S., Verkest, D., et al.: DRESC: A retargetable compiler for coarse-grained reconfigurable architectures, *International Conference on Field Programmable Technology* (*FPT2002*), pp.155–173 (2002).
5) Toi, T., Nakamura, N., Kato, Y., et al.: High-Level Synthesis Challenges and Solutions for a Dynamically Reconfigurable Processor, *The International Conference on Computer-Aided Design* (*ICCAD'06*), pp.702–708 (2006).
6) Tunbunheng, V. and Amano, H.: Black-Diamond: A Retargetable Compiler Using Graph with Configuration Bits for Dynamically Reconfigurable Architectures, *The 14th Workshop on Synthesis and System Integration of Mixed Information technologies* (*SASIMI*), pp.412–419 (2007).
7) Yoshikawa, T.: A dynamically reconfigurable processor for streaming processing, *Tutorial of the ICFPT 2007* (2007).
8) Nishimura, T., Hirai, K., Saito, Y., Nakamura, T., Hasegawa, Y., Tsutsusmi, S., Tunbunheng, V. and Amano, H.: Power Reduction Techniques for Dynamically Reconfigurable Processor Arrays, *18th International Conference on Field Programmable Logic and Application* (*FPL2008*) (2008).
9) Saito, Y., Sano, T., Kato, M., Tanbunheng, V., Yasuda, Y. and Amano, H.: A Real Chip Evaluation of MuCCRA-3: A Low Power Dynamically Reconfigurable Processor Array, *The 2009 World Congress in Computer Science, Computer Engineering and Applied Computing*, pp.283–286 (2009).
10) Sano, T., Saito, Y., Kato, M. and Amano, H.: Fine Grain Partial Reconfiguration for energy saving in Dynamically Reconfigurable Processors, *19th International Conference on Field Programmable Logic and Applications* (*FPL2009*), pp.530–533 (2009).
11) Yamamoto, T., Hironaka, K., Hayakawa, Y., Kimura, M., Amano, H. and Usami, K.: Dynamic VDD Switching Technique and Mapping Optimization in Dynamically Reconfigurable Processor for Efficient Energy Reduction, *7th International Symposium on Applied Reconfigurable Computing* (*ARC2011*) (2011).
12) Chen, D., Cong, J. and Fan, Y.: Low-Power High-Level Synthesis for FPGA Architecture, *Proc. International Symposium on Low Power Electronics and Design*, pp.134–139 (2003).
13) Pandey, R. and Chattopadhyay, S.: Low-Power Technology Mapping for LUT based FPGA: A Genetic Algorithm Approach, *Proc. 16th International Conference on VLSI Design*, p.79 (2003).
14) Kim, J., Yang, H., Ryu, K. and Kim, H.: FPGA Low-Power Technology Mapping for Reuse Module Design under the Time Constraint, *Proc. Future Generation Communication and Networking*, pp.57–61 (2008).
15) Chen, D., Cong, J., Dong, C., Li, F. and Peng, C.: Technology Mapping and Clustering for FPGA Architectures with Dual Supply Voltages, *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, Vol.29, No.11, pp.1709–1722 (2010).
16) Hassan, H., Anis, M. and Elmary, M.: A leakage-aware CAD flow for MTCMOS FPGA architecture, *Proc. 2005 ACM/SIGDA 13th International Symposium on Field Programmable Gate Arrays*, p.267 (2005).
17) Friedman, S., Carroll, A., Essen, B.V., Ylvisaker, B., Ebeling, C. and Hauck, S.: SPR: An architecture-adaptive CGRA mapping tool, *Proc. ACM/SIGDA International Symposium on Field Programmable Gate Arrays* (*FPGA*), pp.191–200 (2009).
18) Tunbunheng, V., Suzuki, M. and Amano, H.: RoMultiC: Fast and Simple Configuration Data Multicasting Scheme for Coarse Grain Reconfigurable Devices, *International Conference on Field Programmable Technology* (*FPT2005*), pp.129–136 (2005).
19) Kleinhans, J.M., Sigl, G., Johannes, F.M. and Antreich, K.J.: GORDIAN: VLSI Placement by Quadratic Programming and Slicing Optimization, *IEEE Trans. CAD*, pp.356–365 (1991).

20) Breuer, M.A.: Min-Cut Placement, *Journal of Design Automation and Fault Trelance*, pp.343–362 (1977).

(平成 23 年 1 月 28 日受付)
(平成 23 年 6 月 15 日採録)

木村　優之
2011 年慶應義塾大学大学院理工学研究科開放環境科学専攻前期博士課程修了．現在，ルネサスエレクトロニクス（株）所属．

弘中　和衛（学生会員）
現在，慶應義塾大学大学院理工学研究科開放環境科学専攻前期修士課程にて動的再構成アーキテクチャの研究に従事．

天野　英晴（正会員）
1986 年慶應義塾大学大学院理工学研究科電気工学専攻博士課程修了．博士（工学）．現在，慶應義塾大学理工学部情報工学科教授．計算機アーキテクチャの研究に従事．