計算機アーキテクチャ 149－6
（２００２．８．２２）

粗粒度再構成型 PARS アーキテクチャのためのコンパイラ

後藤義人† 谷川一哉† 児島彰‡ 弘中哲夫‡

† 広島市立大学大学院 情報科学研究科 情報工学専攻
‡ 広島市立大学 情報科学部

従来の再構成型アーキテクチャはハードウェアのサイズを越えるプログラムは実行不可能という問題点があった．それを解決するために我々はPARSアーキテクチャを提案している．PARSアーキテクチャではプログラムをハードウェア資源に合わせて分割し，それを動的に切り替えることにより，実際のハードウェアよりも大きなサイズのプログラムを実行することが可能である．PARSアーキテクチャでプログラムを実行するにはアプリケーションをデータフローで記述し，アーキテクチャで実行できる形に変換しなければならないが，現段階ではこれを手作業で行っている．本稿ではデータフローを４つ組で表現したコードをPARS用のアセンブラ記述に変換するコンパイラについて述べ，暗号プログラムFEALのFK関数を用いて評価を行った結果を示す．

Development of the Compiler for PARS Architecture, a Coarse Grain Reconfigurable Architecture

Yoshito Goto†, Kazuya Tanigawa†, Akira Kojima‡, Tetsuo Hironaka‡

†Graduate School of Information Sciences, Hiroshima City University
‡Faculty of Information Sciences, Hiroshima City University

PARS Architecture is a coarce grain reconfigurable computer which can reconfig function units and wires every cycle. PARS Architecture can execute large scale programs which is over the limit of hardware size by dividing the program into fixed size, called “page”, and swich pages dynamically. But handwork is needed to implement programs on PARS architecture, and it is inefficient. It is necessary to develop a compiler for the PARS architecture. In this paper, we show the design and implementation of the compiler for PARS architecture which translate intermediate code into assembler code for PARS architecture. Compilation results of a test program using FK function in encryption program FEAL are also shown.

1 はじめに

近年，メディア処理やデータ通信処理に対する要求が高まってきている．これらの処理ではデータサイズが可変であること，高速な処理，様々な演算器構成などが要求される．しかし，既存のプログラミングモデル，アーキテクチャではこれらの要求に対応することが難しくなってきている．そこで，これらの要求を満たす再構成型コンピュータが注目されている [1].

再構成型コンピュータはアプリケーション毎に最適なビット幅や演算器構成を実現することで，従来のシステムに比べ高速な処理を可能とする．しかし，アプリケーションの規模がシステムのハードウェア量よりも大きい場合，全く計算不可能になってしまうという問題点があった．

そこで我々は対象となる問題をハードウェア資源の制限に従って固定のサイズに分割し，それらを動的に切り替えて処理を行うことができるPARSアーキテクチャを提案している [2][3].

現段階ではプログラミング環境が整っておらず，アプリケーションを実行するまでに膨大な手間と時間を消費してしまう．現在，PARSASと呼ばれる専用のGUIツールを用いたプログラミングを行っているが，配線やレジスタ割り付けなどの処理を手動で行わなければならないため大規模なプログラムをマッピングすることが非常に困難である．そこで本研究ではPARS用のコンパイラを現在開発している．

3節ではPARSアーキテクチャについて説明し，4節ではPARS用のコンパイラについて説明する．5節ではバックエンドのアルゴリズムについて説明し，6節で評価を行う．最後に7節でまとめる．

2 関連研究

PARSアーキテクチャに関連する研究として再構成型コンピュータであるREMARC[4]やRaPiD[5]などが挙げられる．REMARCはメインのRISCプロセッサのコプロセッサとして動作する再構成コンピュータである．REMARCのプログラミング環境においてユーザーはREMARCで実行する命令を明示的に記述したC言語とREMARCで実行される命

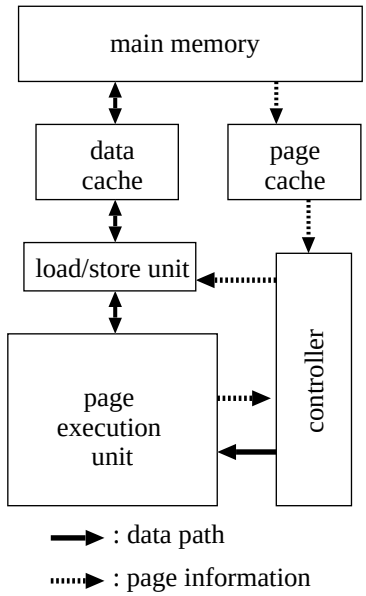

図 1: PARS アーキテクチャの構成

令の構成情報を記述する必要がある． RaPiD は深い計算パイプラインを構成することができる粗粒度再構成型アーキテクチャであるが，これも配線数や演算器数の制限によりハードウェアに依存したプログラミングモデルになっているため，ユーザーはハードウェアに関する専門的な知識が必要となるといった問題点が挙げられる．

3 PARS アーキテクチャ

PARS アーキテクチャは動的に再構成を行うことにより実ハードウェアより大きいサイズのプログラムでも実行可能とする再構成型コンピュータである． PARS アーキテクチャでは対象となるアプリケーションをデータフロー及びコントロールフローで表現した記述を， PARS アーキテクチャが 1 サイクルで実行可能な命令に分割し，レジスタ情報や配線情報を加えたページと呼ばれる単位を動的に切り替えることにより実行する． PARS アーキテクチャの特徴として以下の 2 つが挙げられる．

- 演算器をベースとする再構成ユニットを持つ
- 1 クロックで動的な再構成が可能

演算器をベースとすることにより LUT （Look Up Table）をベースとする場合よりも再構成情報を削減することができ，その結果として 1 クロックでの動的な再構成を実現可能にしている．

アーキテクチャの構成図を図 1に示す．構成図において実線の矢印がデータの流れ，点線の矢印が構成情報の流れを表している．図中の Page Execution Unit が PARS アーキテクチャの再構成部にあたる．

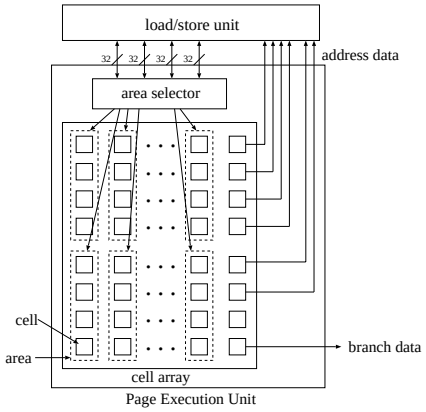

図 2: 再構成部の構成

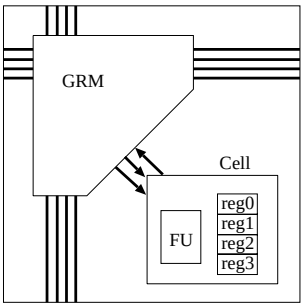

図 3: セルと GRM の構成

3.1 再構成部の構成

PARS アーキテクチャに基づいたプロトタイププロセッサ tempo では，再構成部は図 2のようになっている．この再構成部は図のように 2 次元アレイ上に演算を行うセルが配置された構造になっている．セルアレイの一番右の列は主にメモリアドレスの生成や分岐判定のために用いられ，通常の演算を行うセルは $8 \times 8 = 64$ 個のセルが存在する．セルでは 8bit の演算を行う．メモリアクセス命令は図のように 4 つのセルをまとめたエリアと呼ばれる単位で 32bit のデータに同時にアクセスする．

3.2 セルと GRM の構成

セルと GRM の構成図を図 3に示す．セルの内部には演算を行なう FU （Function Unit）と 4 つのレジスタから構成される． FU は 8bit のデータに対して算術演算，論理演算，シフト演算，マルチプレクサの演算を行う． GRM(Global Routing Matrix) はセル間に繋がれた配線のスイッチングを行うユニットである．

4 PARS 用コンパイラ

4.1 アプリケーションのマッピング方法

PARS アーキテクチャでのアプリケーションのマッピングは以下のような手順で行われる．

1. 対象とするアプリケーションをデータフローグラフ，コントロールフローグラフの形で表現する
2. データフローグラフをノードの深さに応じて分割する．
3. ハードウェア資源の制約などに従って一度の再構成で実行可能な命令列であるページに分割する．
4. 各ページに存在するデータや演算の割り付けを行う．
5. 割り付けられたデータに従って配線の情報を付加する．

通常のプログラミングと異なる点として，同時に実行可能な命令列を区別することや，実際のハードウェアでの演算を行うユニットの指定，配線の指定などを行わなければならないことなどが挙げられる．

従来は，これらの手順のうち 1 〜 3 を机上で行い，4，5 の演算とデータの割り付け，配線の 2 つの処理は PARSAS と呼ばれる専用の GUI ツールを用いて設定を行っていた．これらの作業には PARS アーキテクチャに関する知識が必要であることや，プログラムの記述に膨大な手間と時間が費やされるという問題点があった．

PARS 用コンパイラは入力となる言語で記述されたアプリケーションを受け取り，1 〜 5 の処理を自動で行なうことを目的とする．

4.2 PARS 用コンパイラの構成

PARS の目指すコンパイラの構成を図 4に示す．PARS アーキテクチャへの入力となる言語はユーザーの使いやすさを考慮すると C 言語などの従来の逐次型言語を用いることが望ましい．しかし PARS アーキテクチャのハードウェアを効率的に使用することを考慮すると，並列性を記述しやすい言語が要求される．よって PARS アーキテクチャでは高水準言語と専用の高並列言語の両方からの入力を可能とするコンパイラの構築を目指す．

高並列言語の仕様は検討中だが，Verilog-HDL 記述のブロッキング代入文，ノンブロッキング代入文のような表現を用いることにより並列性を記述することを検討している．

コード変換のバックエンド側において通常のコード変換と異なる点について説明する．まず並列実行を前提としているため同時実行可能な命令列を分割する処理を行う．PARS アーキテクチャにおいてこの同時実行可能な命令列を FB(Function Block) と呼ぶ．さらにハードウェアの資源の制約によって FB を一度の再構成で実行可能な命令列に分割する．この一度の再構成で実行可能な情報をページと呼ぶ．最後に演算やデータの配置を行う配置処理，配線の設定を行う配線処理を経て PARS 用のアセンブラコードに変換される．

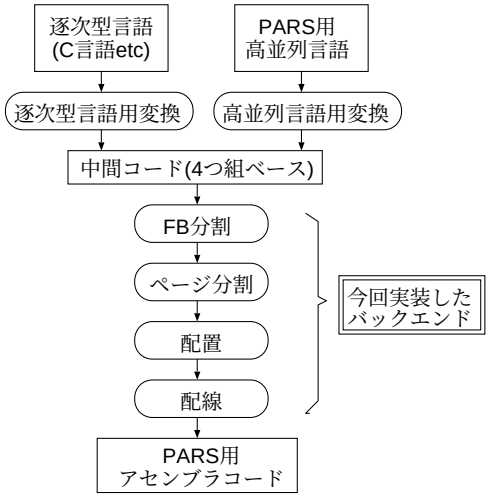

図 4: PARS 用コンパイラの構成

```
//通常の演算
add, temp1, data1, data2

//メモリアクセス命令
ld, [data1,data2,data3,data4], base_ad, offset_ad

//分岐命令
bp, br_data, jp1
```

図 5: 中間言語

PARS アーキテクチャは以上のようなプログラミング環境の構築を目標としているが，全ての部分を一度に設計するのは困難であるため，今回はバックエンドの設計を行った．

5 バックエンドの設計・実装

バックエンドでは以下のように演算をノードとするデータフローグラフを 4 つ組ベースで表現したものを中間言語を入力とし，それを PARS アーキテクチャで実行可能なアセンブラコードに変換することを目的とする．中間言語の例を図 5に示す．

命令の表現方法は以下の通りになっている．

- 通常の演算
  （命令，デスティネーション，ソース 1，ソース 2）
- メモリアクセス命令
  （メモリアクセス命令，データ，ベースアドレス，オフセットアドレス）
- 分岐命令
  （分岐命令，分岐判定データ，分岐先ラベル）

また中間言語で用いるデータは全て 8bit 単位に分割されており，命令も PARS アーキテクチャで実行可能な命令に変換されていることを前提とする．ここで用いられるデータはコンパイラによってレジスタまたは配線に割り付けられる．

バックエンドのおおまかな処理の流れは以下の順序で行われる．

1. FB 分割処理
   まず入力コードを基本ブロックに分け，演算をノードとするデータフローグラフで表現し，真の依存に従って同時実行が可能な命令列に分割する．
2. ページ分割処理
   同時実行可能な命令列をさらに PARS アーキテクチャの再構成部の資源量（FU 数，メモリアクセスポート数）の制限に従って分割する．
3. 配置処理
   命令とデータを再構成部の FU とレジスタに割り付ける．
4. 配線処理
   再構成部に割り付けられたデータにしたがって，セル間の配線を担う GRM の設定を行う．

以下，各処理についてそれぞれ説明する．

5.1 FB 分割処理

FB 分割処理では入力コードを真の依存にしたがって同時実行が可能な命令列に分割する処理を行う．ここで実行可能な時刻が複数存在するノードに関しては最も早く実行が可能な時刻にノードを割り付けるものとする． FB 分割の処理手順を以下に示す．

1. 基本ブロックの最初から存在するデータを深さ０とし，各命令に深さの情報を設定する
2. ４つ組のコードにおいて各ソースについて調べ，深さの数値を比較
3. 二つのソースのうちより大きい値＋１をその４つ組の深さとする
4. 基本ブロックが終るまで繰り返し
5. 同じ深さに命令を並べ換え，命令列を分割する．

5.2 ページ分割処理

ページ分割処理では同時実行が可能な命令列である FB を，さらにハードウェアの資源の制約によって分割を行う．再構成部において FU 数が 64 個，メモリアクセスポートが 4 つであることから， 1 ページで実行可能な演算は最大で 64 命令，メモリアクセス命令は最大で 4 命令に制限される． FB 内の演算数やメモリアクセスの回数がハードウェア資源の制約を満たしている場合は FB がそのままページとなる．

ページ分割の手順を以下に示す．

1. 命令カウンタ，メモリアクセスカウンタを設ける．
2. コードを上から走査
3. コードが通常の命令だった場合命令カウンタ＋1，メモリアクセス命令だった場合は命令カウンタ＋4，メモリアクセスカウンタ＋１
4. もし命令カウンタが 64 もしくはメモリカウンタが 4 の時，ページ分割を行う．

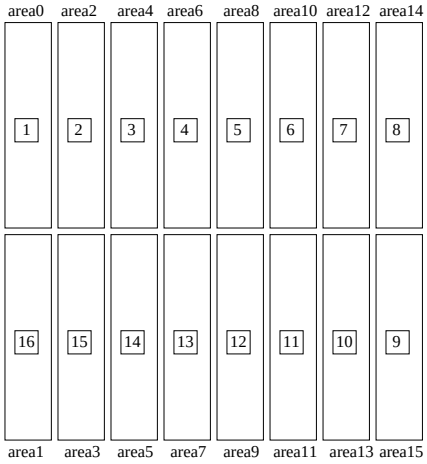

図 6: 再構成部へのデータ配置順序

5. 2 〜 4 を全ての FB に対して実行する．

5.3 配置処理

配置処理では各ページで実行される演算と演算に用いられるデータを実際の再構成部に割り付ける処理を行う．割り付けるのは命令とデータだが，データの割り付け方によって大きく性能が変わってしまうことが考えられるため，データの割り付けを行った後，命令の割り付けを行う．

データを割り付ける場合，最初にエリアの選択を行う．エリアの選択では配線長が比較的短くなる経験的手法を用いて，図 6のような番号順でデータを配置する．

次にエリア内の各セルに存在するレジスタにデータの割りつけを行う．ここで考慮しなければならない特別な事項として， PARS アーキテクチャではメモリアクセス命令はエリア単位で行われ，また各セルの同レジスタ番号にのみアクセスを行うという制約がある．このため図 7において，番号の順にデータの割り付けを行うものとする．

またキャリーは隣接するセルにしか伝搬できない構造のため，キャリーを用いた演算は必ず隣合うセルに演算を配置することが必要となる．

以上を考慮して，データの配置を以下の順番で行った後，命令の割り付け処理を行うものとする．

1. 各ページのメモリアクセス命令に用いるデータを配置
2. 各ページのキャリーを行うデータを配置
3. 各ページの通常の演算に用いるデータを配置
4. 割り付けられたデータをデスティネーションに持つ命令を FU に割り付ける．

5.4 配線処理

配線処理では実際に割り付けられたデータにしたがって各ページに配線情報を付加する処理を行う．セル間の配線は GRM のスイッチを切り替えることにより実現している． GRM は東西南北にそれぞれ

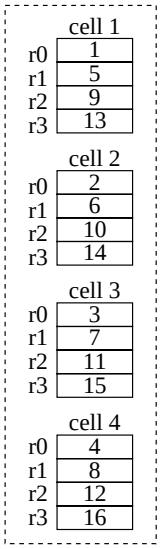

図 7: エリア内のデータ配置順序

4 つの出力を持っており，GRM に入力される配線は入力方向以外の全ての方向に出力することができる．今回はアルゴリズムの簡単化のため，最短経路へのパスが使用不可能だった場合には迂回は行わず，転送が不可能だったデータを転送するだけのページを挿入することにより解決する方法を採用した．

ここで東西南北の方向はそれぞれ e,w,s,n で表し，配線番号は 0 〜 3 で表すものとする．例として図 8のようにセル A からセル B への配線を行う場合，付加される GRM の情報は以下の 3 つである．

1. データを出力するセルに隣接した GRM の出力方向とレジスタ番号
2. 出力したデータの方向転換を行う GRM の入力の方向と配線番号及び出力方向
3. データを入力するセルに隣接した GRM の入力方向と配線番号

5.5 PARS 用アセンブラコード

中間言語を入力として FB 分割，ページ分割，配置，配線の処理を行った結果として出力される PARS 用アセンブラコードの例を図 9に示す．

PARS 用アセンブラコードでの配置配線などの記述について説明する．

- cell[番号]
  その命令が実行されるセル番号を表す
- r0 〜 r3
  演算に使用するレジスタ番号を表す
- page 命令列
  1 度の再構成で実行される命令列を表す
- grm(方角, 番号)
  セルへの入力となる方向，配線番号を表す
- grm[番号](方角, レジスタ)
  配線の設定を行う GRM の番号，GRM を出力する方角，出力するレジスタを表す

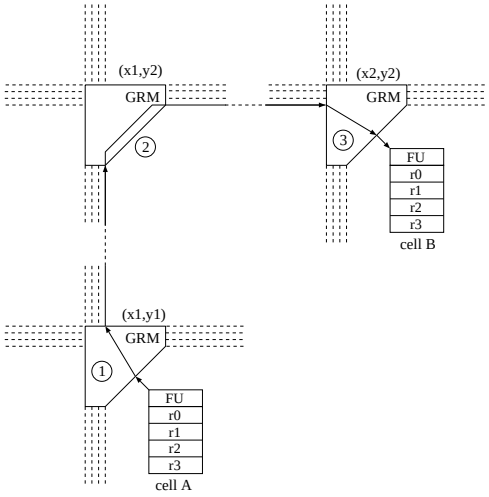

図 8: 必要となる配線情報

```
page 1{
cell[9].add    r0.grm(n,3),r0
grm[8]         (s,r0)
〜
}
page 2{
〜
```

図 9: PARS 用アセンブラコード

6 評価

提案したバックエンドの手法によりコード変換を行い，PARS 用アセンブラコードを実際に実行した結果と，手作業によるプログラミングとの比較を行った．サイクル数，使用レジスタ数，使用レジスタ率の項目について評価を行った．

6.1 アプリケーション

評価アプリケーションには暗号プログラム FEAL[6] の FK 関数を用いた．FEAL における多くの処理は 8 ビットで行なわれるため，粗粒度型である PARS アーキテクチャへのマッピングが容易であるという特徴がある．FK 関数は C 言語で 17 行に相当し，この部分を Sun WorkShop 6 の C コンパイラでコンパイルした結果 (cc -fast)，アセンブラ命令で 74 命令であった．

6.2 結果

評価の結果を表 1に示す．評価の結果，提案手法は手作業と同じページ数でコード変換を行うことができた．この結果から提案手法でのコード変換でも従来の手作業と同じ速度で実行が可能であることがわかる．使用レジスタ数が手作業の場合よりも増加しているのは，現段階ではレジスタの再割り付けを

表 1: サイクル数とレジスタの使用量

	手作業	提案手法
ページ数 (サイクル数)	15	15
使用レジスタ数 (個)	8	32
使用レジスタ率	3%	12%

行っていないことが要因だと考えられる．よって今後レジスタの再割り付け処理を行うことによりこの差を減少させる見込みは十分あると言える．

7 まとめ

本稿ではPARSアーキテクチャ用のコンパイラのバックエンドを設計し，中間コードからPARS用のアセンブラコードを生成するアルゴリズムを示した．暗号プログラムFEALのFK関数を用いて評価を行った結果，提案手法は従来の手作業と同等の処理速度が達成できることが見込めた．今後の課題としてレジスタがあふれた場合にスピルコードを挿入するアルゴリズムや，各アルゴリズムの最適化が必要である．その上でより体系的な評価を行っていく予定である．

謝辞

本研究を進めるにあたり，貴重な御意見を頂いた広島市立大学情報科学部の北村俊明教授に感謝致します．本研究の一部は武田計測先端知財団の協力のもとで行われた．

参考文献

[1] 末吉 敏則, “Reconfigurable Computing System の現状と課題 - Computer Evolution へ向けて - ”, 信学技報, CPSY96-91, pp. 111-118, 1996.

[2] 谷川 一哉, 弘中 哲夫, 吉田 典可, “PARS アーキテクチャの詳細設計に関する一考察”, 情報処理学会研究報告 2001-ARC-144, pp. 31-36, 2001.

[3] Kazuya Tanigawa, Testuo Hironaka, Akira Kojima, and Noriyoshi Yoshida : “A Generalized Execution Model for Programming on Reconfigurable Architectures and an Architecture Supporting the Model”, http://www.lirmm.fr/fpl2002/home.html, FPL 2002, September 2002 (to be appear).

[4] T.Miyamori, K.Olukotun, “REMARC: Reconfigurable Multimedia array Coprocessor”, IEICE Trans.Inf. & Syst, Vol.E82-D, pp.389-397, February 1999.

[5] C.Ebeling, DC.Cronquist, P.Franklin, C.Fisher, “RaPiD - A Configurable Computing Architecture for Compute-Intensive Applications”, University of Washington Department of Computer Science & Engineering Tech Report TR-96-11-03.

[6] 清水 明宏, 宮口 庄司, “高速データ暗号アルゴリズム FEAL”, 電子情報通信学会論文誌, Vol. J70-D, No. 7, pp. 1413-1423, July 1987.