RC-007

無線信号処理向けアレイプロセッサにおける データ列変換向け命令情報生成方法

Automatic Code Generation for ReOrdering Unit on Array Processor for Baseband Signal Processing

石原 希実† 小堀 友義† 関 克敏† 池川 将夫†
Nozomi Ishihara Tomoyoshi Kobori Katutoshi Seki Masao Ikekawa

１．はじめに

昨今では大容量データ通信の要求から無線通信方式のスループット向上の研究が盛んである. これに伴い, 無線通信用の信号処理が複雑になってきており, これを高速に実行するアーキテクチャが求められるようになった. さらに, 近年では様々な無線通信方式が規格化されており, 無線通信機器はこれら複数の方式に柔軟に対応する必要がある. そのため, これら複数の無線通信方式に柔軟に対応することができ, かつ,信号処理を高速に実行できる Application Specific Processor(ASIP)が求められている.

このような要求にこたえるべく, 我々は Multiple Input Multiple Output – Orthogonal Frequency Division Multiplexing (MIMO-OFDM) 処理に特化したリコンフィギュラブルシストリックアレイプロセッサ CORSAEngine[1]を開発した. CORSAEngine はマルチバンクのメモリを持ち, メモリから読み出した複数のデータを高並列なシストリックアレイ演算部において並列に処理することによって, 高い演算性能を実現している. ここで, CORSAEngine はシストリックアレイ演算部における効率的な演算を実現するために, メモリから読み出したデータを演算に適した並びに並び替える ReOrdering Unit (ROU) をメモリ-シストリックアレイ演算部間に持つ[2]. ROU はユーザーが作成したプログラムで制御が可能であり, 実行する処理毎に並び替え方を変えることにより高い柔軟性を獲得している.

しかし, ユーザーが ROU 向けのプログラムを作成する際には, ROU に入力されるデータの時間/ポートの並びを全て把握し, かつ, アーキテクチャが持つ複数の制御ユニットの動きをサイクル毎に把握しなければならない. このため, ROU にはプログラム作成が困難であるという問題がある. そこで本稿では ROU がもつ制御ユニットへの制御情報を, ROU に入力, 出力されるデータの並びを示した情報 (データフォーマット)から自動生成するアルゴリズムを提案する.

2. 関連研究

複数の演算ユニットを持つプロセッサにおける, 演算ユニットへのデータ供給方法はそのプロセッサのアーキテクチャによって異なる. 例として RICA[3]は C 言語等高級言語による記述を可能とするために RISC 型 CPU に近い制御方法を用いている。具体的には, RICA は複数の演算ユニットへのデータ供給を行うために複数のメモリを持っているが, 各々のメモリは RISC 型 CPU の命令セットが備えるようなロード/ストア命令によってデータを入出力する. これによって, 与えられたコードのデータ依存関係制約の分析と, 各々の演算ユニットへの処理割り振り以外は, 通常のコンパイラと同様の方法で機械語を生成すればよいため, コンパイラの実装が容易となる利点がある.

一方, 無線信号処理や画像処理等特定の領域における信号処理では, 演算時のデータの並びが予め決まっている場合が多い. この特性を利用し, ある制御信号に従ってデータの入出力に必要な一連のアドレスをメモリに供給する, もしくは読み出した(書き込む)データに対して一定の並び替えを行う構成を採用し, 命令発行等に必要なオーバーヘッドを削減するアーキテクチャがある. このようなアーキテクチャの例としては RaPID[4]のアドレス生成機構や KressArray[5] の Generic Address Generator (GAG)[6]があげられる. RaPID[4]のアドレス生成機構は初期値, オフセット, 繰り返し数によって制御され, ある制御信号をトリガとして, 初期値にオフセットを加算する処理を繰り返し数だけ行い, これら処理で生成される一連のアドレスを 1 つのメモリに対するアドレスとして使用する. 一方 Kress Array の GAG は RaPID と類似の構成であるが, RaPID の繰り返し数, オフセットがそれぞれ 1 つであるのに対し, GAG は複数ループ, オフセットをサポートする. すなわち, 2 重もしくは 3 重ループを用いてアドレスの出力を行う. これによってメモリアクセスに必要な処理をプロセッサに代わって行うことができる. また, 文献[7]は, メモリアクセスに関わる様々なアーキテクチャの比較検討を行っている.

CORSAEngine ROU[2]は内部にマルチバンクメモリと複数の GAG を持ち, データの並び替えを行う. ここで, KressArray では Xputer[8]を用いて GAG への制御情報を自動で生成する手法が説明されている. しかし, KressArray アーキテクチャは基本的にシングルメモリをターゲットとしているのに対し, CORSAEngine はマルチバンクメモリを持ち, 複数の GAG を用いて並列にアクセスを行うため, 同一の方法を適用することはできない. そこで本稿ではマルチバンクメモリ, 複数 GAG 環境における制御情報生成手法を提案する.

† 日本電気株式会社

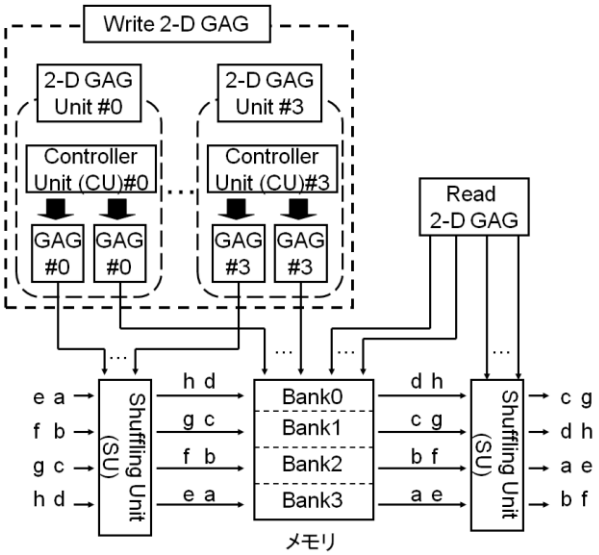

図 1　ROU アーキテクチャ概要

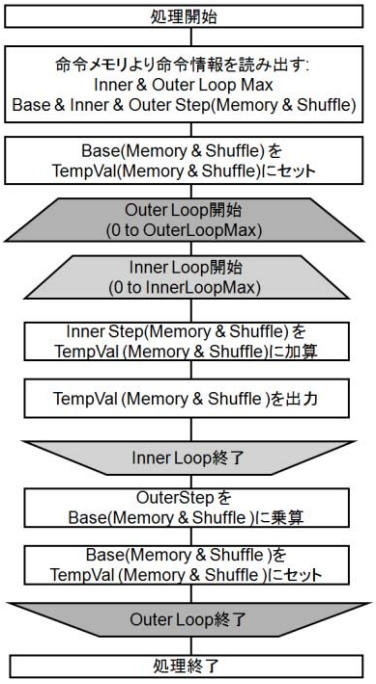

図 2 2-D GAG の制御方法

3.　ROU アーキテクチャ[2]

3.1　概要

ROU アーキテクチャの概要を図 1 に示す. ROU は 4 ポートの入出力を持つユニットであり, 内部に入力されたデータ同士を入れ替えて出力する Shuffling Unit(SU) , 1 入力/1 出力のメモリ 4 バンク及びこれらを制御するための Write 2-Dimentional(2-D) GAG, Read 2-D GAG を持つ.

本アーキテクチャの動作について, 図 1 に示したデータ a～h の並べ替え例を用いて説明する. 入力されたデータに対して, まず SU がポート間の入れ替えを行う.図 1 では同時に入力された a,b,c,d のポートを入れ替え, d,c,b,a として出力する様子を示している. 次に,これら出力データに対し, メモリへの書き込み, 読み出し処理を通じて時刻間の順序を入れ替える. 図 1 では a～h の時刻順序が入れ替わり, 後の時刻に入力された h,g,f,e が先に出力される様子を示している. 最後に, メモリから出力されたデータに対し, SU がポート間の入れ替えを行う. 図 1 では h,g,f,e として入力されたデータが g,h,e,f として出力する様子を示している. なお, これら処理の制御は Write(Read) 2-D GAG が行う. 具体的には ROU への入力からメモリへの書き込みまでを Write 2-D GAG が制 御し, メモリからの読み出しから出力までを Read 2-D GAG が制御する.

3.2　Write/Read 2-D GAG

SU とメモリへの入出力を制御する Write(Read) 2-D GAG はそれぞれが同一の構造を採用している. 以下では図 1 の Write 2-D GAG 構造を参照して説明する.

Write 2-D GAG は内部に複数の 2-D GAG ユニットを持つ. 各ユニットの内部には 2 つの GAG[6]と, これらを制御する 1 つの Controller Unit(CU)がある. 2 つの GAG はそれぞれが SU とメモリへ制御情報を生成する.

SU への制御情報は, 各ポートに対してデータを取得して出力するポート番号を示すものである. 例として, 図 1 の Write 2-D GAG が制御する SU について説明する. a, b, c, d が入力されるポート番号をそれぞれ 0, 1, 2, 3 とすると, これらに対する制御情報はそれぞれ, 3, 2, 1, 0 となる. 一方, メモリへの制御情報はデータを書き込むメモリワードアドレスを示すものである.

これら 2 つの GAG を CU が制御する方法について図 2 に示す.図 2 は CU が 1 つの命令情報に従って 2 つの GAG を制御するフローを示している. なお,図 2 において用いる要素名を以下に列挙する.

- Inner & Outer Loop Max : 内側(外側)ループの回数
- Base(Memory & Shuffle) : メモリ(シャッフル)ベース値
- Inner & Outer Step(Memory & Shuffle):
  内側(外側)ループメモリ(シャッフル)ステップ値
- TempVal(Memory & Shufle) :
  メモリ(シャッフル)用一時変数

図 2 の通り, CU は 2 重ループを使って, 2 つの GAG の制御を実行する. CU は 2 つの GAG の制御を開始すると, まず 1 つの制御情報を命令メモリから読み出す. ここで, この制御情報は上記に列挙した要素のうち, TempVal 以外の要素で構成される. 次に読み出した Base(Memory & Shuffle)を一時変数 TempVal(Memory & Shuffle)に格納し, 内側ループを実行する.

内側ループでは TempVal(Memory & Shuffle)を Inner Step(Memory & Shuffle)によって更新し, 更新した値を出力する. 内側ループ処理が終了したら, Base(Memory & Shuffle)を Outer Step(Memory & Shuffle)で更新し, これを次の内側ループ用のベース値として TempVal(Memory & Shuffle)にセットする. 外側ループが終了したら, 本制御情報の実行終了である. なお, 後続の制御情報がある場合には, さらに後続の制御情報について本フローを実行する.

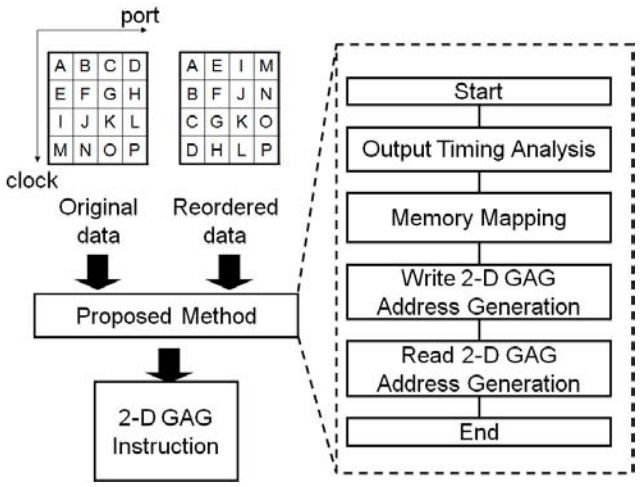

図 3 Proposed Algorithm

4. 命令情報の自動生成手法

2-D GAG の命令情報を生成する提案手法への入出力概要と, 手法のフローを図 3 に示す. 提案手法は ReOrdering Unit への入力(Original Data)と並べ替え結果出力(Reordered data)の様子を 2 次元配列として示した入出力データフォーマットを用いて 2-D GAG 命令情報を生成する. なお, 入出力データフォーマットの横軸はポート, 縦軸はクロックを示している.

提案手法は 4 ステップで構成される. 具体的には, 入力されるデータと出力されるデータを比較し, メモリマッピングを決定するための分析を行う Output Timing Analysis, その結果を使ってメモリマッピングを決定する Memory Mapping, マッピングの結果を用いて Write(Read) 2-D GAG の制御情報を生成する Write(Read) 2-D GAG Address Generation からなる.

4.1 Output Timing Analysis

本処理では, ROU に入力されるデータと, ROU からの出力データを比較し, メモリマッピングを行うための参照データを作成する. 本処理の概要を図 4 に示す. 図 4 に示すとおり, ROU への入出力データ列は clock 数 x port 数サイズの 2 次元配列であり, 入力を Original data, 出力を Reordered data として示している. この 2 つのデータ配列から clock 数行 x port 数列の Output timing data を作成する. Output timing data は, Original data の各要素に対し, ROU から出力される際に出力開始を先頭として数えたサイクル数を示したものである. 例として, 黒の点線で示した A というデータは, Reordered data を参照すると,出力開始 1 サイクル目に ROU から出力されると分かる. よって, Output timing data の A の位置に 1 を記述する.

4.2 Memory Mapping

Output Timing Analysis が終了したら, 次にメモリマッピングデータを作成する. 本処理の様子を図 5 に示す.図 5 のとおり, 本処理は Output timing data に示された出力

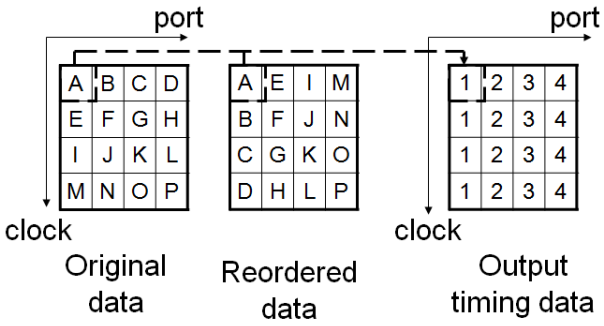

図 4 Output Timing Analysis

サイクル数が早い順に, Original data の各要素に対してメモリの配置バンク, ワードアドレスを決定する. なお, 本処理では各メモリバンクが 1 入力/1 出力であるというアーキテクチャ上の制約を考慮して行う. これはすなわち, 同時刻に出力するデータは別メモリバンクに配置しなければならないということを意味するため, 本処理はこの制約に違反しないようにメモリマッピングを行う.

図 5(a)は 1 サイクル目に出力するデータ(A, E, I, M)のマッピング方法を示している. まず, 入力が最も早い A のマッピングについて考えると, A はどのバンクに配置しても上記制約に違反することはない. そこで入力されたポートと同一のバンクに配置する. 次に E のマッピングを考える. このとき, A と同一のバンクに配置すると, A と同時刻に出力できない. そこでバンクを 1 ずらし A と隣接するバンクに配置する. I, M についても同様にマッピングする.

次に 2 サイクル目に出力するデータ(B, F, J, N)のマッピングを行う(図 5(b)参照). まず, B については入力ポートと同一のバンクに配置しても上記制約に違反しないため, 同一バンクに配置する. 一方, F のデータにおいては入力ポートと同一バンクに配置すると上記制約に違反する. そこでバンクを 1 ずらして配置する. J, N についても同様にマッピングする.

上記を繰り返し, Output timing data の全データの解析が終了したら, 本処理を終了する. この状態を図 5(c)に示す.

4.3 Write 2-D GAG Instruction Generation

メモリマッピングの作成が終了したら, Write 2-D GAG の制御情報を生成する. 本処理ではまず, Write 2-D GAG が生成すべき SU 及びメモリへの制御情報列を作成する. この様子を図 6 に示す.図 6 の Address data は Original data の各要素に対して Write 2-D GAG が生成すべき制御情報について（SU 用制御情報, メモリ用制御情報）の形式で表わしている. 例として, データ A は Memory mapping の 1 バンク目, 1 ワード目に書き込むべきデータである. そこで, Address data には(1,1)と記述する.

次に 2-D GAG への命令情報を作成する. このフローは入力ポート毎に行う. 1 つの入力ポートに対する制御情報生成フローを図 7 に示す. ここで, 図 7 の Addr(n)は対象としたポートの n クロック目の情報を示す. また, 図 7 で使

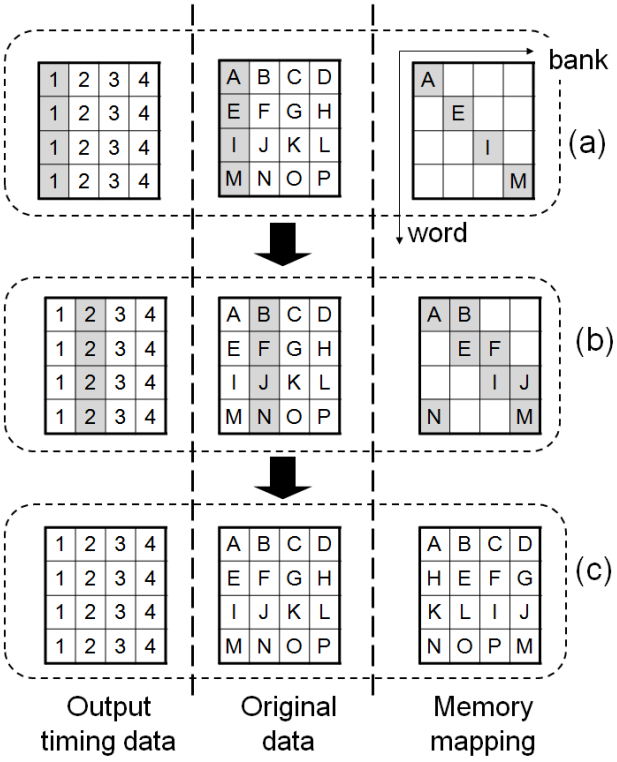

図 5 Memory Mapping

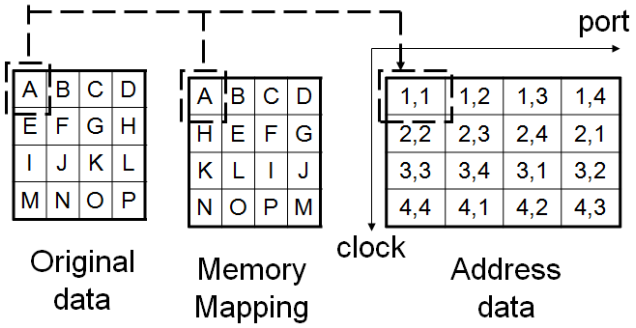

図 6 Address Generation

用する要素名は基本的に図 2 と同様の要素を用いるが, Memory & Shuffle については M & S と短縮して示している.

命令情報生成は 1 つの入力ポートの全データが分析し終わるまで繰り返し行う. 1 つの制御情報を生成するフローは主に, 初期値準備, Inner Loop Analysis, Outer Loop Analysis の 3 つから構成される.

初期値準備では, Base, Inner Step, Inner Loop Max の設定を行う. 具体的には, 分析を開始した n クロック目のデータを Base 値としてセットし, さらに n+1 クロック目のデータとの差分を計算して Inner Step にセットする. また, Inner Loop Max = 0 とセットする.

次に, Inner Loop の分析を行う. ここでは, 上記で計算した Inner Step を用いて出力可能なアドレスのクロック数

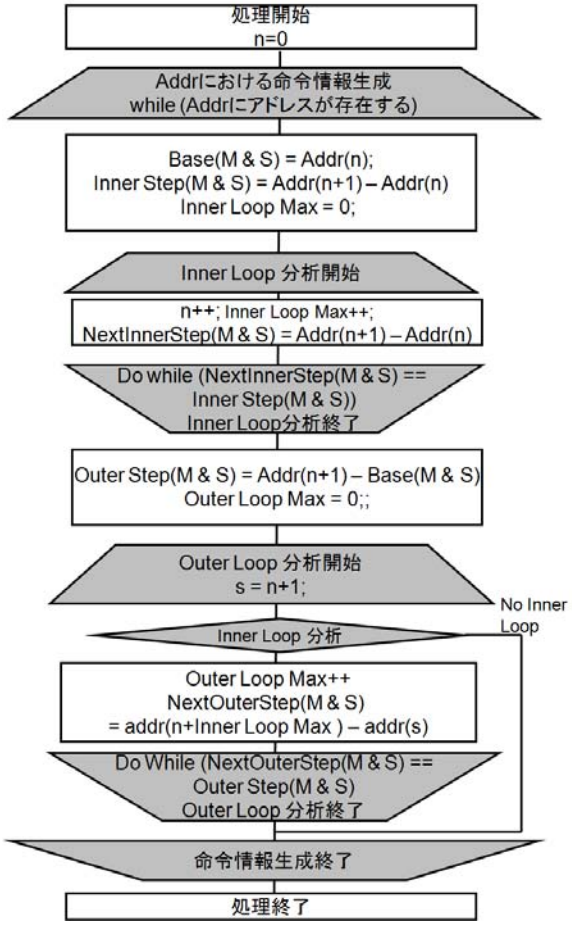

図 7 Write 2-D GAG Instruction Generation

を計算する. 具体的には, n クロック目のデータと n+1 クロック目のデータの差分を計算し, それが Inner Step と一致する場合, Inner Loop Max に 1 を加える. 一致しない場合は本処理を終了し, 次の Outer Loop 分析に移る.

Outer Loop 分析では, まず Outer Step をセットする. 具体的には, Inner Loop 分析が終了したクロックのアドレスと Base の差分を Oute Step としてセットする. 次に, その直後のサイクルから Inner Loop が成立する区間が存在するか, すなわち, 上記で求めた Inner Step と Inner Loop Max を使って出力できるアドレス区間が存在するかどうかを分析する. Inner Loop が成立する区間が存在した場合, Outer Loop Max を 1 加算する. さらに, その後のアドレス区間が Outer Loop を用いて出力可能であるかを次のように分析する. まず, Outer Loop 分析を開始した時点のクロックを s, Outer Loop 分析において Inner Loop が成立する区間があると判明した時点のクロックを m とする. このとき, m+1 クロック目のアドレスが, s クロック目のアドレスに Outer Step を加算したアドレスと一致する場合, さらなる Outer Loop が成り立つ可能性があると判断し, 再び, Outer Loop 分析を行う. 一方, Inner Loop で表せない制御情報が存在した場合や, Outer Loop オフセットが一致しなかった場合は, 分析を終了する. なお, 生成が終了した時点で, Addr にアドレスが残っている場合, 新たな制御情報生成フローを行う.

4.4 Read 2-D GAG Instruction Generation

Write 2-D GAG の命令情報生成が完了したら, 最後に Read 2-D GAG の命令情報を生成する. ここで, 本処理は, 入力データが Memory mapping と, Reordered data となる以外は, Write 2-D GAG 制御情報生成フローと同様である.

5. 性能評価

本手法の評価を行う. 評価では本手法により生成された命令情報と手設計で最適化した場合の命令情報を実現するために必要な命令情報量を比較して, 本手法の性能について述べる.

5.1 命令実行サイクル数

MIMO-OFDM 復調処理に必要な FFT 等ソフトウェアを CORSAEngine に実装した際の所要サイクル数について, 提案手法により生成された命令情報及び, 最適化済みの命令情報との比較結果を表 1 に示す. 表 1 左列は評価に使用したソフトウェアである. なお, 重み付加算と Clip については 1200 ワードのデータ処理におけるサイクル数である.

表 1 より, 最適化済み命令情報と提案手法によって生成された命令情報における処理サイクル数は同じである. 以上より本提案手法は, 最適化された命令列とサイクル数の点において同等の性能となる命令を生成することができるといえる.

5.2 命令情報量

次にソフトウェアの所要命令情報量について提案手法により生成された命令情報及び, 最適化済み命令情報との比較結果を表 2 に示す. なお, 表 2 には提案手法による命令情報と最適化済み命令情報を比較した際の増加率も示した.

FFT/IFFT では提案手法の結果は最適化結果と比べて 0.3%多い. これは, FFT/IFFT の回転子等パラメタ設定部において, 一度のパラメタ設定で複数ポイントの FFT/IFFT を処理するために不規則かつ複雑な並び替えを行っており, 最適化版ではこれに対応するために緻密な制御を行っているからである. 一方, 重みつき加算や Clip 処理では最適化コードと同等のメモリ量を実現している. 命令情報量の総計を見ると, 提案手法の結果は最適化済みのものと比べて増加率は 0.15%である. 以上より提案手法は, 一部高度な最適化が必要となる制御方法以外は, 最適化された命令情報と同等のメモリ量を実現することができるといえる.

5.3 命令実行サイクル数と命令情報量の考察

命令情報量と命令実行サイクルの関係について以下に考察を行う. 上記結果より, 本提案手法は FFT/IFFT において最適化済みの結果と比べて命令情報量は 0.3%増加するが, サイクル数は同一である. これは, 命令のデコードに必要なサイクル数が並べ替えデータ全体の入力に必要な時間を下回るためである. すなわち, N サイクル分のデータの並べ替えを行う際に, 並べ替えの命令の読み出し, デコードに M サイクル必要であるとすると, $N \geqq M$ の場合, ソフトウェア実行サイクル数は N サイクルとなる. FFT/IFFT はこの関係を満たしているため, 命令実行サイクル数は同一となる.

表 1 所要サイクル数

処理	ROU (最適化)	ROU (提案)	CROU (最適化)
32pt FFT	364	364	355
2048pt FFT	2636	2636	2627
重み付加算	1364	1364	1355
Clip	1420	1420	1411

表 2 命令メモリ量

処理	ROU (最適化)	ROU (提案)	増加率 (%)	CROU (最適化)
FFT/IFFT	21645	21717	0.3	45704
重み付加算, Clip	4401	4401	0	1244
総計	26046	26118	0.15	46948

6. 比較評価

文献[2]では, 本稿が扱う ROU 以外に, 複数のメモリバンクから読み出された, または複数メモリバンクへ書き込まれる一連のデータに対し, 遅延器によって時間方向の並びを変え, さらに Shuffling Unit(SU)によってポート間の並びを変えることによって, データ並べ替えを行うという別の構成方法 (従来 ROU) が示されている. また, 評価において, 本稿が扱う ROU を適用した際には, 従来 ROU よりもメモリ使用効率が 12%向上することが示されている. 一方, あるソフトウェアを実装した際の所要命令情報量や所要処理サイクル数に関する評価は, 文献[2]では述べられていない. そこで, 本稿が扱う ROU と従来 ROU において CORSAEngine 向けソフトウェアを実装した際の所要サイクル数と命令情報量を分析し, 各々の ROU のソフトウェア処理上の特性を明らかにした上で, 提案手法の有効性について検証する. なお, 以下では文献[2]に述べられている従来 ROU を, 区別のために Conventional ROU (CROU)と示すこととする.

6.1 ソフトウェア実行サイクル数

表 1 右に CORSAEngine に CROU を適用し, 最適化済み命令情報を用いた場合の所要サイクル数について示した.

表 1 を参照すると, 実行サイクル数は CROU と比較して ROU が多い. これは ROU においては GAG 等の処理準備をするための初期化が必要なためである. しかし, この

サイクル数は 10 サイクル未満であるため, 全体の処理サイクル数への影響は少ない. よって, ROU は CROU と同等の性能を有しているといえる.

6.2 命令情報量

表 2 の増加率の右に CROU を適用した場合の最適化済み命令情報量を示した。表 2 より ROU の命令情報量は, FFT/IFFT については CROU より少なく, その他のライブラリは多いことが分かる.

本稿の FFT/IFFT は Cooley-Turkey アルゴリズムを用いており, 比較的小さい点数の FFT/IFFT を組合せて大きな点数の FFT/IFFT を実現している. ここで, 上記アルゴリズムでは 2 つの FFT/IFFT 点数処理間でデータの並べ替えを行うが, この並べ替えを CROU 及び ROU が処理する. この並べ替えは規則的ではあるものの, 大きな点数の FFT/IFFT では規則の周期が長くなる. この場合, ROU は 2 重ループ構造によって長い周期の並べ替えを少ない命令情報量で実現できる. これに対し, CROU では 1 つの周期の全てのデータに別々の命令情報を割り振る必要があるため, 命令情報量が大きくなる.

一方, その他のライブラリは並べ替え規則の周期が短い. この場合は, CROU, ROU とも生成する命令情報数自体は少なくなる. 従って, 命令情報量に対して 1 つの命令情報に必要なビット数の影響が大きくなるが, この 1 つの命令情報に必要なビット数は ROU のほうが多い. 具体例を挙げると, CROU/ROU に入力されたデータをそのままスルーして出力する処理を考えた場合, 必要となる命令情報量は CROU が 72 ビットであるのに対し, ROU は 180 ビットであり, CROU と比べて 2.5 倍の命令情報が必要となる. しかしこういったライブラリは FFT/IFFT に比べれば命令情報量自体が少なく, 全体として占める割合は小さい. 実際, 総計としては ROU のほうが CROU よりも命令情報量は少ない. よって, ROU のほうが必要となる命令情報量は少なくなるといえる.

7. まとめ

本稿では CORSAEngine のデータ並べ替え機構である ROU に対し, その入力と出力を 2 次元の配列で表した入出力データフォーマットから, ROU に対する命令情報を自動で生成する手法を提案した. 提案手法は入出力データフォーマットから ROU 内部のメモリマッピングを自動で決定し, 決定したメモリマッピングに従って, ROU の Write(Read) 2-D GAG への命令を自動で生成する. 本手法によって, 複雑な 2-D GAG 命令情報を直接記述するのではなく, 所望の並べ替えを入出力データフォーマットとして記述するだけでよいため, ROU における並べ替えを容易に実現できる. 評価においては, CORSAEngine ソフトウェアライブラリを用いて生成される命令情報量とソフトウェア実行に必要なサイクル数について, CROU と比較しながら分析した. 評価の結果, ROU は CROU とほぼ同等の処理性能を有し, 必要となる命令情報量は少ないことが分かった. また提案手法が生成した命令は最適化した命令と比較すると, 命令情報の増加量が 0.15%程度であることが分かった. 一方, 処理サイクル数においては最適化した命令と同じサイクル数でソフトウェアを実行できることが分かった. 以上より, 本提案手法は命令情報量, 処理サイクル数の観点から有効な命令を生成できるといえる.

文 献

[1] K.Seki, T.Kobori, J.Okello and M.Ikekawa, “A Cordic-Based Reconfigurable Systolic Array Processor for MIMO-OFDM Wireless Communications,” IEEE Workshop on Signal Processing Systems, pp.639-644, Oct. 2007.

[2] 小堀友義, 石原希実, 関克敏, 池川将夫, “アレイプロセッサ向けプログラマブルデータ並び替えユニットの実現”, リコンフィギャラブルシステム研究会, 2011 年掲載予定

[3] Sami. Khawam, Ioannis Nousias, Mark Milward, Ying Yi, Mark Muir, Tughrul Arslan, “The Reconfigurable Instruction Cell Array,” IEEE Trans. on Very Large Scale Integration Systems, Vol. 16, No. 1, pp.75-85, Jan. 2008.

[4] D.C. Cronquist, C. Fisher, M. Figueroa, P. Franklin, and C. Ebeling, “Architecture Design of Reconfigurable Pipelined Datapaths,” Proc. Conf. Advanced Research in VLSI, pp. 23-40, 1999.

[5] R. Hartenstein, M. Herz, T. Hoffmann, U. Nageldinger, “Using KressArray for Reconfigurable Computing,” Proc the International Society for Optical Engineering (SPIE) Vol. 3526, Conference on Configurable Computing: Technology and Applications, 1998

[6] R. Hartenstein et al,.“A novel sequencer hardware for application specific computing”, Application-Specific Systems, Architectures and Processors, pp.392-401, July, 1997.

[7] M. Herz, R. Hartenstein, M. Miranda, E. Brockmeyer, F. Catthoor, “Memory Addressing Organization for Stream-Based Reconfigurable Computing,” IEEE Proc. Conf. Electronics, Circuits and Systems, pp.813-817, vol.2, 2002

[8] R. Hartenstein, M. Herz, T. Hoffmann, U. Nageldinger, "Exploiting Contemporary Techniques in Reconfigurable Accelerators," Proc. 8th International Workshop on Field-Probrammable Logic and Applications, pp.189-198, 1998