メディア処理向け動的再構成可能LSIの自動コード生成手法

Method of Automatic Program Code Generation for Dynamically Reconfigurable LSI

黒田　亮　　松崎　秀則　　浅野　滋博
■ KURODA Akira　■ MATSUZAKI Hidenori　■ ASANO Shigehiro

動的再構成可能なLSIは，回路を動作させながらソフトウェアによってその構成を変更することができるLSIである。
東芝は，メディア処理を主な用途とする独自の動的再構成可能LSIアーキテクチャ“FlexSword$_{TM}$”を開発している。このLSIを活用するには，処理内容を記述したソースプログラムから，LSIの動作を規定するプログラムコード（機械語命令に相当）を自動で生成するコンパイラが必要である。当社はプログラムコード生成のために解決すべき問題を階層的にグローバルな問題といくつかのローカルな問題に分割することで，現実的な時間内に最適化されたコードを生成できるようにする手法を確立した。

Dynamically reconfigurable large-scale integrations (LSIs) are constructed with the capability to change a logic function while it is being executed.
Toshiba has developed the FlexSword$_{TM}$ dynamically reconfigurable LSI for media processing. To utilize this LSI, a compiler that generates program code (executable binary code) automatically from high-level program language is desirable. We have established a novel compiling method that combines a global optimization technique and some local optimization techniques to generate optimized code in a reasonable time.

1　まえがき

デジタルテレビや携帯電話などでは，受信した動画像の復号や画質改善などに様々なメディア処理技術が用いられている。このようなメディア処理を実現するLSIに対しては，高性能と低消費電力が求められている。これらの要求に応えるため，固定のハードウェア回路で構成された専用のLSIが用いられてきた。

ところが，様々な画質改善処理などがLSIの付加価値となるにつれて，ソフトウェアによって内部の回路構成を変更可能な柔軟性（プログラマビリティ）が求められるようになってきた。プログラマビリティがあると，様々な要求を単一のLSIで満たすことができるため再利用が可能となり，LSIの開発コストが削減できる。

高性能，低消費電力，及びプログラマビリティのすべての要求を満たすLSIとして，動的再構成可能LSIがある。東芝は，パイプライン型動的再構成方式を用いた，メディア処理向けLSIアーキテクチャ“FlexSword$_{TM}$”を開発した[(1), (2)]。

一般に，LSIの動作を規定するプログラムコードを手作業で記述することは困難であるため，従来のLSIには，C言語などのソースプログラムからLSI専用のプログラムコードを自動で生成するコンパイラが存在する。ここでは，FlexSword$_{TM}$のアーキテクチャに適用するプログラムコードの生成手法と，新しく確立したコンパイル手法について述べる。

2　動的再構成可能LSIアーキテクチャ FlexSword$_{TM}$の概要

当社が開発した動的再構成可能LSIアーキテクチャであるFlexSword$_{TM}$の構成を**図1**に示す。

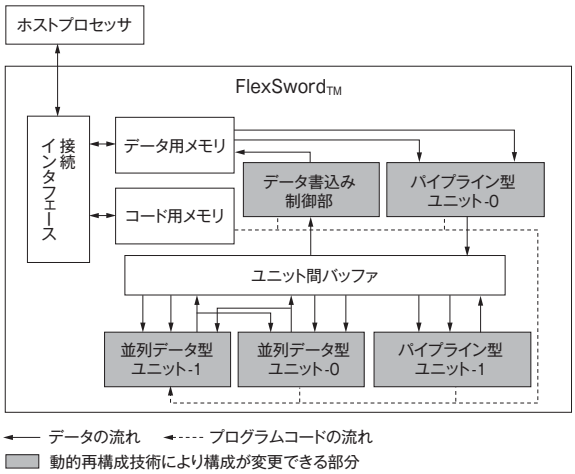

図1. FlexSword$_{TM}$の構成 ― 動的再構成を行う複数のユニットを持っている。
Configuration of FlexSword$_{TM}$ dynamically reconfigurable LSI

2.1　FlexSword$_{TM}$の構成

FlexSword$_{TM}$の構成は，データ用メモリ，コード用メモリ，ホストプロセッサとの接続インタフェース，パイプライン型ユ

ニット，並列データ型ユニット，ユニット間バッファ，及びデータ書込み制御部から成る。各々のユニットとデータ書込み制御部は，動的再構成技術により1回の演算を処理するたびに構成が変更できる。

データ用メモリ及びコード用メモリは，ホストプロセッサから転送されるデータとプログラムコードをそれぞれ格納するメモリである。

ユニット間バッファはユニット間のデータ転送に用いられるメモリである。ユニットにより書き込まれた演算結果データを，そのデータを用いるすべての演算が完了するまで保持し，完了した時点で消去する。

データ書込み制御部は，演算結果をデータ用メモリに書き込むための制御を行う。

パイプライン型ユニットは，**図2**に示すように，8列×5行の格子状に配置された40個のALU（Arithmetic and Logic Unit：演算論理装置）と，ALU間接続を実現する四つのシャッフルユニットで構成される。この構成は，メディア処理の特徴に合うようにALUの配置やALU間の接続が工夫されている。例えば，Row-0のALUの演算結果は，次行Row-1の複数のALUに出力でき，そのデータ転送経路はシャッフルユニットで実現される。ただし，ハードウェアリソースを節約するため，実現可能なデータ転送経路は限定されている。

並列データ型ユニットは八つのALUで構成され，八つのデータを同時に演算できる。ただし，パイプライン型ユニットとは異なり，これらのALUは同時に1種類の演算しか実行できない。また，ALU間のデータ転送は行わないSIMD（Single Instruction Multiple Data）ユニットである。

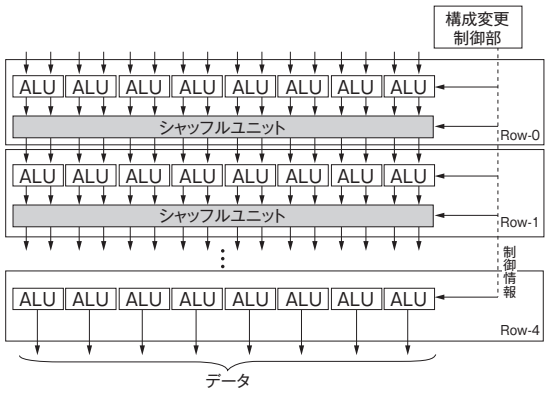

図2．パイプライン型ユニットの構成 ― 8×5個のALUがシャッフルユニットにより接続されている。

Configuration of pipeline-style processing

2.2 FlexSword$_{TM}$の動作

ホストプロセッサは，データとプログラムコードを転送し，処理開始命令を通知することでFlexSword$_{TM}$の動作を開始させ，処理された演算結果をデータ用メモリから受け取ることでFlexSword$_{TM}$の動作を終了させる。

FlexSword$_{TM}$は，パイプライン型ユニット - 0を利用してデータ用メモリからデータを読み出して演算を開始し，最終的にはデータ書込み制御部を介してその演算結果をデータ用メモリに書き戻す。その過程で，ほかのユニットを任意の順序や回数で経由しながら演算処理を進めることができる。なお各ユニット間のデータの受け渡しは，ユニット間バッファ経由で行われる。

3 コンパイルの概要

ここでは，コンパイラのフローについて述べる。

既存のコンパイラは，一つの演算装置を持つプロセッサ，複数の演算装置を持つ並列プロセッサ，及びALUを格子状に配置した構成のLSIを対象としたコンパイル技術を適用している[3], [4]。

一方，FlexSword$_{TM}$は，パイプライン型ユニットや並列データ型ユニットなどの異種の再構成可能なユニットを複数持っているため，上記のような既存コンパイル技術を適用できない。そこで，異種の再構成可能なユニットを持つ動的再構成LSI向けのコンパイル技術を新たに確立した。

コンパイラは**図3**に示すように，データ依存解析，クラスタリング，配置・配線，構成変更スケジューリング[5]の四つのフェーズから成る。対象処理をC言語で記述したソースプログラムを入力とし，FlexSword$_{TM}$用のプログラムコードを出力とする。

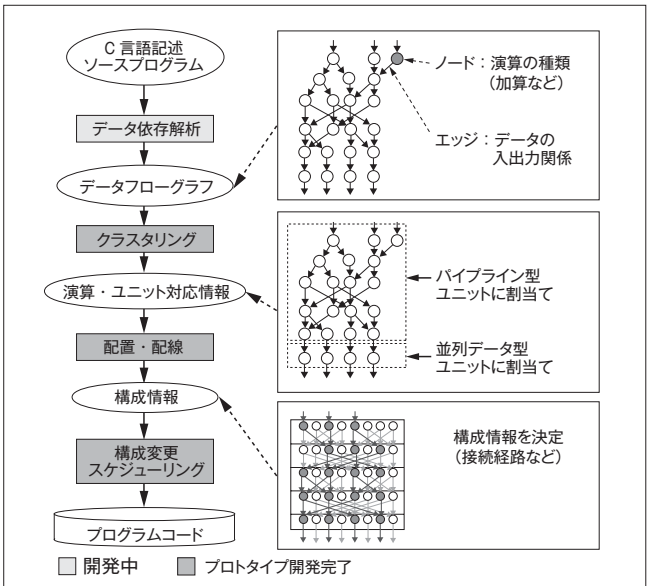

図3．コードコンパイルフロー ― C言語で記述されたソースプログラムから，構成情報や構成変更の順序とタイミングを自動決定し，プログラムコードを生成する。

Flow of code compiling for FlexSword$_{TM}$

データ依存解析では，C言語で記述されたソースプログラムを解析し，解析結果をデータフローグラフとして生成する。データフローグラフは図3に示すように，演算をノード，演算間のデータ依存関係をエッジとして記述したものである。

次に，クラスタリングでは，上記のデータフローグラフを解析して各演算を処理するのにもっとも適したユニットを決定し，演算・ユニット対応情報を生成する。

配置・配線では，ユニット内部において各演算を実行するALUとALU間のデータ転送経路を決定し，構成情報を生成する。

最後に構成変更スケジューリングでは，構成情報とデータ依存解析で得られた演算間のデータ依存関係を考慮して，演算実行のために構成情報の変更順序とそのタイミングを決定し，プログラムコードとして出力する。

現在，クラスタリング部，配置・配線部及び構成変更スケジューリング部はプロトタイプが完成している。残るデータ依存解析部は，解析可能なソースプログラムの範囲を拡大しながら開発中である。

4　自動コード生成手法

ここでは，コードコンパイルフローの重要なフェーズであるクラスタリング，配置・配線，及び構成変更スケジューリングについて**図4**を用いて説明する。

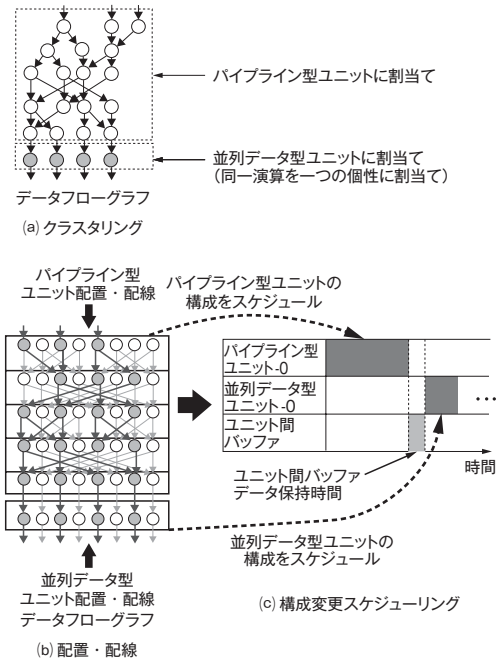

図4．提案手法を用いたプログラムコード生成例 — クラスタリング，配置・配線，及び構成変更スケジューリングの例を示す。

Example of program code generation using proposed method

4.1　クラスタリング

クラスタリングでは，図4(a)に示すように，データフローグラフ内の演算をもっとも適当と思われるユニットに割り当てる。この段階ではユニットの数が無限にあると仮定し，物理的制約は考慮しない。これは，各々のユニットが実行時に構成を変更できるためである。

ユニットを効率よく使用するためには，より多くの演算を一つのユニットに割り当てることが望ましい。FlexSword$_{TM}$は2種類の性質の異なるユニットを持っており，これらを効率よく利用するために，各ユニットの特長に適した演算群を割り当てる必要がある。パイプライン型ユニットに適した演算群は，データ依存関係がある演算群である。また，並列データ型ユニットに適した演算群は，同一演算を行う演算群である。

しかしながら，単に一つのユニットに多くの演算を割り当てるだけでは，FlexSword$_{TM}$全体の演算効率を高くすることはできない。これは異種のユニットが存在するために，片方のユニットに対する効率の最大化が他方のユニットに対する効率の最大化に寄与するとは限らないためである。このため，このコンパイラのクラスタリングにおいては，複数のユニット内ALUの利用効率を考慮した割当てアルゴリズムを採用し，全体的な最適化を実現した。

図4(a)では，データ依存関係がある演算群をパイプライン型ユニットに，四つの同一演算を並列データ型ユニットに割り当てている。

4.2　配置・配線

図4(a)においてユニットに割り当てられた演算群の配置・配線の結果を図4(b)に示す。パイプライン型ユニットの配置・配線では，演算のALU割当てとデータ転送経路の決定を行う。また，並列データ型ユニットの配置・配線では，演算群のALU割当てを行う。

パイプライン型ユニットでは，各ALUが実行する演算及びALU間の接続関係について膨大な組合せの構成が実現できる。一方でハードウェア資源の節約のため，ALU間のデータ転送経路などに数多くの制約が存在する。このため，多くの場合において，この膨大な組合せの中で対象演算群を実現できる構成は非常に限られたものとなり，これを短時間で発見することは難しい。例えば，すべての構成を探索する全探索方式は，探索領域が非常に大きいため，現実的な時間では解を求めることができない。かりにパイプライン型ユニットのすべての構成を探索する場合，必要な計算時間は数十年間単位であることがわかっている。そこで，このコンパイラにおける配置・配線では，演算間のデータ依存関係から明らかに実現できない構成をあらかじめ探索対象から除外していくことで，短時間での探索を実現している。

一方，並列データ型ユニットの配置・配線は，可能な組合せが少なく，演算に用いる演算間バッファ上データの配置から容

易に決定することができる。

4.3 構成変更スケジューリング

構成変更スケジューリングでは，これまで無限としていたユニットの数を物理的なユニットで実現する順序と，ユニット間バッファ利用タイミングの決定を行う。ユニット間バッファの数は上限があるので，その制約を超えないように順序とタイミングを決定することが重要である。

ユニット間バッファに一度書き込まれたデータは，このデータを使用するすべての処理が完了するまで保持され続け，完了した時点で消去される。このため，いったん書き込まれたデータがなかなか使用されないといったデータの長期間保持は，ほかの演算結果を書き込むためのユニット間バッファ数の減少につながる。更に，データの長期保持が重なると，書込み可能なユニット間バッファが存在しない要因となる。これを回避するため，ユニット間バッファに保存されたデータを可能なかぎり早く使用するようにスケジューリングを行っている。

スケジューリング結果を図4(c)に示す。図では縦軸がユニットとユニット間バッファを表し，横軸が時間を表している。図4(c)の例では，まずパイプライン型ユニットが動作し，演算結果をユニット間バッファに格納する。次に並列データ型ユニットがそのデータを用いて処理を行うスケジュールとなっている。

5 あとがき

メディア処理に特化した動的再構成LSIアーキテクチャFlexSword™向けに，C言語で記述されたソースプログラムから自動でプログラムコードを生成するためのコンパイル手法を確立した。この手法では，プログラムコード生成のために解決すべき問題を，階層的にグローバルな問題（クラスタリング）とローカルな問題（配置・配線及び構成変更スケジューリング）に分割することで，現実的な時間内に最適化されたコード生成を実現している。

今後の課題は，LSIアーキテクチャの特長を生かし，より高性能な処理を実現するプログラムコードを生成できるようにすることである。様々なプログラムに対するコンパイル結果を評価しながら，各フェーズにおける最適化精度がいっそう向上するように改良していく。

文 献

(1) Yoshikawa, T., et al. "An Implementation of Hardware Accelerator using Dynamically Reconfigurable Architecture". HotChips. Stanford University, USA, 2006-08, IEEE. available from＜http://www.hotchips.org/archives/hc18/2_Mon/HC18.S4/HC18.S4T3.pdf＞, (accessed 2008-11-10).

(2) Yamada, Y., et al. "Implementation and Evaluation of the Processor for Stream Multimedia Applications using Dynamic Reconfiguration". COOLChips X. Yokohama, 2007-04, IEEE. p.325－349.

(3) Mei, B., et al. "DRESC：A retargetable compiler for coarse-grained reconfigurable architectures". In International Conference on Filed Programmable Technology. The Chinese University of Hong Kong, Hong Kong, 2002-12, IEEE. p.166－173.

(4) Singh, H., et al. Morphosys: an integrated reconfigurable system for data parallel and computation-intensive applications. IEEE Trans. On computers, **59**, 5, 2000, p.465－481.

(5) Kuroda, A., et al. "An Implementation of scheduling algorithm for Dynamically Reconfigurable Architecture using heuristic method". ISICE2007. Japan, 2007-09, IEEE. p.92－97.

黒田　　亮　KURODA Akira
研究開発センター　コンピュータネットワークラボラトリー。システムLSIの研究・開発に従事。
Computer & Network Systems Lab.

松崎　秀則　MATSUZAKI Hidenori
研究開発センター　コンピュータネットワークラボラトリー研究主務。システムLSIの研究・開発に従事。
Computer & Network Systems Lab.

浅野　滋博　ASANO Shigehiro
研究開発センター　コンピュータネットワークラボラトリー研究主幹。システムLSIの研究・開発に従事。IEEE会員。
Computer & Network Systems Lab.