Vol.2009-ARC-184 No.16
2009/8/5

複数プロセッサによるパイプライン型動的再構成エンジンFlexSwordの共有方式

徳永将之[†1]　山田裕[†1]
吉川宜史[†1]　浅野滋博[†1]

FlexSword は複数のメディア処理をプログラマブルに実行可能なアクセラレータである．複数の動的再構成可能な演算ユニットがパイプライン的に動作することでメディア処理を効率的に実行可能である．本稿は，ホストプロセッサが複数存在するときの FlexSword の利用方式についての検討を行ったものである．1 つの FlexSword を複数のホストプロセッサで共有することで，FlexSword のアイドル時間を削減し，面積あたりの性能を高めることができる．H.264 動画像デコードを対象アプリケーションとして FlexSword を共有したときの性能及び面積の変化について FlexSword 非共有の場合と比較し，共有ホストプロセッサ数が 4 のとき最大約 1.7 倍の面積効率の向上を確認した．

Resource Sharing Mechanism of Reconfigurable Engine FlexSword for Multiprocessor System.

Masayuki Tokunaga ,[†1] Yutaka Yamada ,[†1]
Takashi Yoshikawa [†1] and Shigehiro Asano [†1]

FlexSwrod is the acceletrator for multimedia peocessing. FlexSword can execute several multimedia applications bacause of dynamic reconfigurable structure. In uni-processor system, FlexSword drive less time than the host processor. For time and area effeciency, multi processor share one FlexSword. Although FlexSword is a powerful accelerator, its performance is limited on uni-processor system because the host processor is not powerful enough to utilize it. Therefore resource sharing of FlexSword on multi-processor system is required. In this paper, we propose the sharing mechanism of FlexSword for multiprocessor system. We enhance the area-efficiency by sharing the FlexSword with multi processors. We evaluate the area-efficiency in simulation and get the result of up to 70% enhancement of area-efficiency when four processors share one FlexSword.

1. はじめに

近年，携帯機器ではより複雑，かつ多様な処理が行われるようになっている．動画像処理についても，従来は固定機器で利用されてきた複雑な処理を携帯機器においても利用したいという要求が高まっている．そのため，携帯機器に搭載されるシステム LSI の高性能化が強く求められている．また，携帯機器においてはバッテリー持続時間の長大化に対する要求も強く，携帯機器向けのシステム LSI は高性能であると同時に低消費電力であることが求められている．以上のような要求に応えるため，それぞれの動画像処理を行う専用の LSI を設計することが一般的である．

一方，動画像処理も多様化しており，動画像圧縮方式を挙げても MPEG-2，H.264/AVC, VC-1, Divx などの様々な方式が利用されている．これらのアルゴリズムは基本的な構造では類似点があるものの細部では多くの差異があり，このような処理一つ一つに対して専用 LSI を設計することは設計コストや開発期間の増大を招く．

上記のような問題に対する解決手段の一つとして専用 LSI をプログラマブルな構造とし，単一のハードウェアで複数の動画像処理に対応することが挙げられる．我々のグループでは動的再構成技術を用いて複数の動画像処理をプログラマブルに実行可能なアクセラレータである FlexSword (Felxible Software oriented Dynamically Reconfigurable Engine) の検討を行っている[1)2)]．FlexSword は複数の再構成可能な演算ユニットを持ち，それらの演算ユニット上でデータをパイプライン的に処理する．FlexSword をアクセラレータとしてプロセッサに接続して用いる事で，複雑なメディア処理を高速かつ低消費電力に実行できる．

本稿では，FlexSword をマルチコア環境において利用するための手法を検討する．各ホストプロセッサがそれぞれ個別の FlexSword と接続するのではなく，複数のホストプロセッサが 1 つの FlexSword を共有する形で接続し，各ホストプロセッサが個別の FlexSword を保持する場合に近い性能を，より小さい面積で実現する．共有モデルでは FlexSword があるホストプロセッサからの指示による処理を実行中に，別のホストプロセッサから処理開始の指示が出た場合，その処理が終了するまで次の処理の開始を待つ．そのため共有するホストプロセッサの数や，FlexSword での処理実効時間の長さによっては性能の低下が大きく

†1 東芝研究開発センター
Toshiba Research & Development Center

なる可能性がある．しかし実際には FlexSword の実行時間は，ホストプロセッサの動作時間と比較して非常に短く，複数のプロセッサからの動作命令を 1 つの FlexSword で処理しても性能の低下はそれほど大きくないと考えられる．そのことを確認するため，FlexSword の共有モデルを用いて実際のメディア処理を行った場合の性能の変化について評価した．

また，複数の処理の実行を指示された FlexSword における後続処理の開始までの待機時間を削減する手法の提案を行い，その効果についても併せて評価した．これは FlexSword の処理のスケジューリングが静的に決定される特性を利用したもので，連続して実行する 2 つの処理において先行する処理の終了前に後続の処理を開始することで処理の実行間隔を短縮するものである．

評価の結果，最大で 1.7 倍の面積効率の増加を確認した．この時，1 つの FlexSword を 4 つのホストプロセッサで共有しており，性能の低下は約 5%である．また，処理の重ね合わせのりようによって FlexSword の実行開始までの待機時間を約 50%削減できることを確認した．

本稿の構成を以下に示す．2 章において FlexSword の特徴及びホストプロセッサとの接続モデルについての説明を行う．3 章で FlexSword を複数のホストプロセッサで共有するためのアーキテクチャについて説明し，4 章でその性能評価を行う．5 章で面積についての評価も行い，面積あたりの処理性能を示す面積効率という指標を用いて共有モデルについての評価を行った．

2. FlexSword システムの構成

FlexSword (Flexible Software oriented Dynamiccaly Reconfigurable Engine) は小規模なハードウェア資源で効率的に信号処理を行うことを目的としたプログラマブルなアクセラレータである[1)2)]．FlexSword はホストプロセッサと接続し，ハードウェアエンジンとして動作する．図 1 に FlexSword とホストプロセッサを接続した FlexSword システムについての概要を示す． FlexSword システムは大きく分けてホストプロセッサ，システムメモリ，FlexSword から構成される．ホストプロセッサは制御バスを介して FlexSword に対して指示を発行する．また，ホストプロセッサと FlexSword の間のデータ転送はシステムメモリを介して行われる．FlexSword はホストプロセッサからの指示に従い動作する．本章では FlexSword アーキテクチャの概要，およびホストプロセッサからの制御方法について述べる．

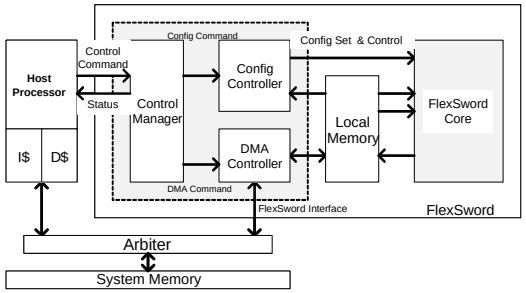

図 1 FlexSword システム概要

2.1 FlexSword

FlexSword はホストプロセッサからの制御命令を処理するインタフェース部，制御情報及び構成情報に従い処理を実行する演算処理部，処理データ及び構成情報を格納するローカルメモリを備える．本稿では演算処理部を FlexSword コア，インタフェース部を FlexSword インタフェース，これらにローカルメモリを加えたものを FlexSword と呼ぶ．

ホストプロセッサからの制御命令は制御バスを介し FlexSword インターフェースで処理される．また，FlexSword は DMA コントローラを保持しシステムメモリとの間で直接データ転送を行う．ホストプロセッサから FlexSword への制御命令は，次のような指示を行う．

- システムメモリから FlexSword のローカルメモリへのデータ転送（DMA 読み込み）
- FlexSword のローカルメモリからシステムメモリへのデータ転送（DMA 書き込み）
- FlexSword コアの構成情報設定
- FlexSword コアの実行開始
- FlexSword 内部の状態確認

2.2 FlexSword コア

図 2 に FlexSword コアのアーキテクチャを示す．FlexSword コアは 5 つの演算ユニットと，演算ユニット間でのデータを受け渡すユニット間バッファ（Inter Unit Buffer, IUB）と備える．ここで 5 つの演算ユニットは Formatter0, Formatter1, AUX0, AUX1, Write Control と呼ばれ， AUX0 と AUX1 は同一のアーキテクチャであり，Formatter0 と Formatter1 についても演算器に関して同一のアーキテクチャからなる．

Formatter0，Formatter1 は，データに対して演算を行う PE (Processing Element) を 5 段備え，パイプライン式に演算を行う．各 PE は 8 並列の ALU を備える．AUX0，AUX1

Vol.2009-ARC-184 No.16
2009/8/5

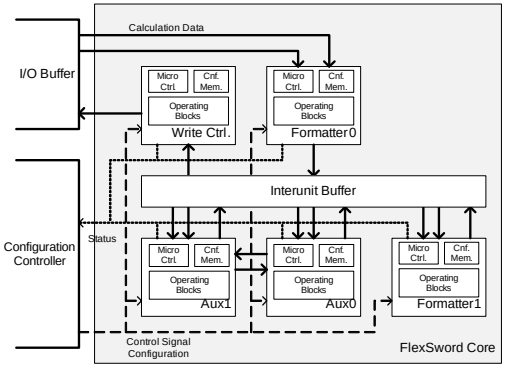

図 2 FlexSword コアアーキテクチャ

表 1 FlexSword 演算ユニットのレイテンシ

演算ユニット	レイテンシ (サイクル)
Formatter 0	9
Formatter 1	6
AUX 0	3
AUX 1	3
Write Control	1

は SIMD 演算器を備える．それぞれの演算ユニットのレイテンシを表 1 に示す． それぞれの演算ユニットは入力データに対して処理を行う実行ユニットと，実行ユニットを制御するマイクロコントローラを持つ．各演算ユニットは個別に構成情報を保持し，独立に動作する．

Formatter 0 は処理に必要なデータをローカルメモリより読み込む．Formatter 0 が読み込んだデータに対して演算を行った後，データは他の演算ユニットへと受け渡され，演算が行われる．Formatter 0 以外の演算ユニットはデータドリブンでの制御を行っており，必要なデータがそろった時点で演算を行い，結果を他の演算ユニットへ受け渡す．規定の演算をすべて終了した後，Write Control よりローカルメモリへとデータを書き戻す．

各演算ユニットが演算を行うタイミング，演算内容，演算結果をどの演算ユニットに対し受け渡すかは構成情報によって静的に決定される．各演算ユニットが保持する構成情報は，コンフィグコードとコントロールコードの 2 種類に分類できる．それぞれの機能については以下の通りである．

- コントロールコード

```
// ローカルメモリに処理データを転送
fs_dma_read( local_adr0, (u32)pix_data    , sizeof(pix_data) , 0 ); // 処理(1)
fs_dma_read( local_adr1, (u32)pix_data2   , sizeof(pix_data2), 0 ); // 処理(2)
func0();

// ローカルメモリに構成情報を転送 (コンフィグ,コントロール)
fs_dma_read  (config_adr, (u32)config_data, sizeof(config_data), 0 ); // 処理(3)
fs_dma_read  (control_adr, (u32)control_data , sizeof(control_data), 0 ); // 処理(4)
func1();

// flexsword core に構成情報を設定 (コンフィグ,コントロール)
fs_load_config (config_adr, 0, 0xf6, 0 ); // 処理(5)
fs_load_control  (control_adr, 0, 8, 2, 2, 0, 4, 0 ); // 処理(6)
func2();

// flexsword core 実行開始
fs_load_init   (end, tag1 ); // 処理(7)

// ローカルメモリから結果を転送
fs_dma_write  (local_adr2, (u32)pix_data     , 0x040, 0, tag1 ); // 処理(8)
func3();

// flexsword の処理が全て完了するまで待機(同期)
fs_wait_halt(); // 処理(9)
```

図 3 FlexSword 実行制御コード例

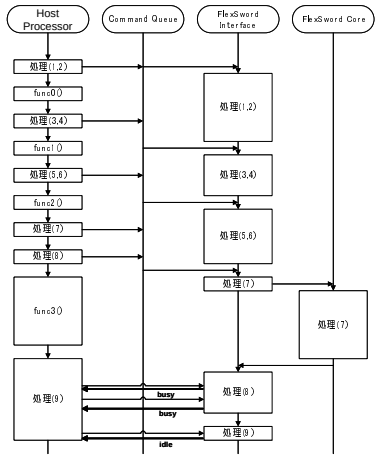

図 4 FlexSword 実行制御タイミングチャート

各演算ユニットが処理を実行するタイミングと，その時の処理内容を指定する ID (コンテキスト ID)，および演算結果の受け渡し先演算ユニットの情報などを保持する

- コンフィグコード
  コンテキスト ID に対応した，それぞれの実行ユニットの処理内容についての情報を保持する

2.3 FlexSword 制御

ホストプロセッサから FlexSword を制御する方法についての説明を行う．ホストプロセッサから制御バスを介して FlexSword に対して制御命令を送信し，処理の開始や構成情報の設定を行う．また，同様に制御バスを介してホストプロセッサは FlexSword コアや制御命令の実行状態を確認することができる．図 3 及び図 4 に処理の手順を示した擬似コードと擬似コード実行の際のタイミングチャートを示す． この例では処理に必要なデータ，構成情報は予めシステムメモリ内に格納されていると仮定する．図 3 の擬似コードではシステムメモリ上のデータをローカルメモリへと転送 (処理 (1，2))， 構成情報のローカルメモリへの転送 (処理 (3，4))FlexSword コアへの設定 (処理 (5，6))，FlexSword コアの処理実行開始処理 (7)，ローカルメモリからシステムメモリへの演算結果の書き戻し (処理 (8))，FlexSword コアとの同期 (処理 (9))，を行う．FlexSword の動作状態を調べ，アイドル状態になるまで待つ．また，ホストプロセッサでは上記の各処理の間に func0, func1, func2, func3 の各関

数を処理する．ホストプロセッサから FlexSword へ入力された命令は FlexSword インタフェース内の命令キューに一時的に保存され，順序関係を保持したまま FlexSword 上で実行される．

3. 複数プロセッサによる共有方式

前章における FlexSword システムは，シングルコア環境での利用を想定している．本章では，マルチコア環境において FlexSword を利用する場合について考える．

マルチコア環境において FlexSword を利用する場合，それぞれのプロセッサが個別に FlexSword を保持するという構造が考えられる．しかし，アプリケーションの実行時には FlexSword の動作時間はホストプロセッサよりも非常に短く，各 FlexSword は必ずしも同時には動作しない．表 2 に FlexSword システム上でアプリケーションを実行した時，処理全体の中で FlexSword が動作している時間の割合を示す． アプリケーションは H.264 動画像

表 2 FlexSword 動作時間の割合 (動作率)

入力画像	FlexSword 動作率 (%)
標準画像”Cycle”	20.3
標準画像”Preakness”	27.6

デコードの信号処理を用いている．どちらの場合も FlexSword の動作時間は全体の 20%以下で，残りの時間はアイドル状態となっている．

従って，1 つのプロセッサに対して 1 つの FlexSword を接続するのではなく，複数のホストプロセッサが 1 つの FlexSword を共有する方式を用いることで，FlexSword の動作率を高め，マルチコア環境において FlexSword を効率よく利用することが出来る．

FlexSword 共有によるデメリットとして，複数のプロセッサから FlexSword の処理開始を指示できることで，複数の処理要求が同時期に発生し，性能の低下が発生することが考えられる．FlexSword を共有するホストプロセッサ数が多い場合，性能の低下はより大きくなる．そのため，FlexSword の共有による面積削減効果と，性能の低下の両方を考慮して FlexSword を共有するホストプロセッサ数の決定を行わなければならない．

本章では FlexSword を複数のホストプロセッサで共有するためのアーキテクチャについての説明を行う．FlexSword 全体ではなく，FlexSword コアのみを共有することで，ホストプロセッサからの制御の複雑化を防いでいる．

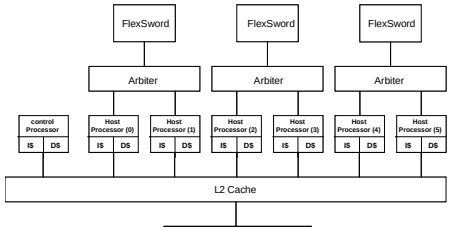

図 5 複数プロセッサによる FlexSword の共有モデル

3.1 FlexSword 共有モデル

複数のプロセッサによる FlexSword 共有モデルの概要を図 5 に示す． 図では 2 つのホストプロセッサが 1 つの FlexSword を共有しているが，3 つ以上のホストプロセッサで共有することも可能である．また，FlexSword に接続してアプリケーションを実行するホストプロセッサとは別に，全体の制御を行う制御プロセッサを持つ．制御プロセッサは，スレッドの生成や実行管理を行う．

FlexSword コアは同時には 1 種類の処理のみを実行するため，時分割での共有を行う．FlexSword コアがあるホストプロセッサからの指示に従い処理を実行しているとき，そのホストプロセッサが FlexSword コアを確保している、と言う．FlexSword コアとホストプロセッサの間にはアービタが存在し，アービタが FlexSword コアを確保するホストプロセッサを決定する．一方，FlexSword インタフェース，およびローカルメモリは各ホストプロセッサごとに個別に存在する．そのため，処理データや構成情報のデータ転送は他のホストプロセッサの状態にかかわらず実行することができる．FlexSword コア実行開始命令はホストプロセッサが FlexSword コアを確保した後に実行される．

アービタは各ホストプロセッサからの FlexSword コア実行要求を受け，FlexSword コアの実行状態の確認を行う．FlexSword コアがどのホストプロセッサにも確保されていない場合，要求したホストプロセッサの確保状態にし，ホストプロセッサに対しては FlexSword コア確保通知を出す．既に別のホストプロセッサが FlexSword コアを確保している場合，発行した実行要求はアービタ内で待機し，FlexSword コアの確保状態が解消されるまで処理の開始を待つ状態となる．

3.2 処理の重ねあわせによる FlexSword コア実行の効率化

1 つの FlexSword コアを多くのホストプロセッサで共有した場合，FlexSword コアを確保するまでの待機時間が増大し，性能が低下する恐れがある．本節では待機時間を削減し，

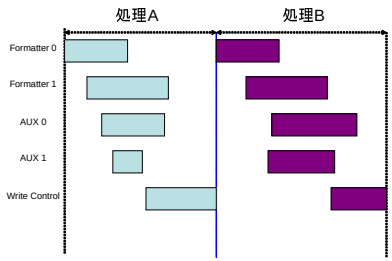

図 6 処理の重ね合わせ (1)

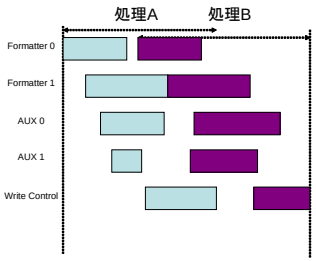

図 7 処理の重ね合わせ (2)

FlexSword 共有による性能低下を抑える手法について述べる．

ここでは，FlexSword コアが複数の処理を連続して実行するとき，その実行間隔を短縮することで FlexSword コアの確保までの待機時間を削減する．

3.2.1 処理の重ね合わせ概要

図 6，図 7 に 2 つの処理を連続して実行した際の FlexSword の動作例を示す．処理 A，処理 B という 2 種類の処理を実行したときの，それぞれの処理における演算ユニットの動作期間を表している． FlexSword では Formatter0 以外の各演算ユニットはデータドリブンの制御を行っている．各演算ユニット有効なデータに対して演算を行い，結果を出力する．そのため各演算ユニットは 1 つの処理において，その開始タイミング，終了タイミングが異なる．その演算ユニットに対し最初の有効なデータが入力した時が開始タイミングとなり，最後の有効な演算結果を出力した時が終了タイミングとなる．また，FlexSword コアで行う処理は構成情報によって静的に決定されるため，各演算ユニットが演算を実行するタイミングはあらかじめ知ることができる．

以上のような FlexSword の特性を用いて，待機時間の短縮を行うことができる．連続して複数の処理を実行する場合，通常は図 6 のように処理 A において全ての演算ユニットが演算を終了してから処理 B を開始する．しかし，処理 A の実行中であっても全ての演算ユニットが動作しているわけではなく，処理 A に必要な演算を全て実行し，待機状態になっている演算ユニットもある．そこで，図 7 のように処理 A が全ての処理を完了する前に処理 B を開始し，処理 B が開始するまでの待機時間を削減する．

3.2.2 実 現 方 法

演算ユニットごとに先行処理の終了後即座に後続処理を開始した場合，先行処理において用いる演算データと後続処理で用いる演算データとが衝突したり，取り違えられて誤った演算結果を出力するなどの問題が発生すると考えられる．そのため，適切な後続処理の開始タイミングを知るための機構が必要となる．

本稿では，あらかじめ構成情報とともにそれぞれの演算ユニットの動作期間を外部から与えることで後続処理の開始タイミングを決定する．FlexSword コア内の演算ユニットの動作時間は構成情報によって静的に決定されるため，事前に動作時間をしることができる．先行処理，後続処理のそれぞれの演算ユニットの動作期間を比較し，先行処理と後続処理が衝突しないタイミングを決定し，後続処理の開始を指示する．

4. 評　　価

これまでに述べた FlexSword 共有モデルについて，シミュレーションによる性能評価を行う．アプリケーションは H.264/AVC デコード処理を対象とし，全処理中の信号処理の主な関数を FlexSword を用いて高速化する．入力画像は標準画像 “Cycle”，または “Preakness” の画像サイズ 720p のそれぞれ 3 フレーム分を用いた．

4.1 評 価 環 境

評価環境として，ホストプロセッサと FlexSword の協調動作を実現するシミュレータを用いた．シミュレータは SystemC 言語によって記述され，サイクルレベルの精度でのシミュレーションを行う．本シミュレータでは FlexSword コアや制御コードの一部に近似を用いている．近似の内容，および近時の利用による影響は以下のとおりである．

FlexSword コア演算の近似

FlexSwod コアの処理について，処理の実行時間の保障は行うが処理内容の保障は行わない．そのため FlexSword が出力するデータについて，値は実際の演算結果とは異なる値を出力する．演算結果は異なるが，その結果のホストプロセッサの動作や FlexSword の他の実行への影響はない．従ってこの近似の利用は性能評価へは影響しない．

構成情報設定命令の近似

FlexSword コアへの制御命令のうち，構成情報の設定命令については本稿ではサポートしない．この近似によって

(1) FlexSword インタフェースによる構成情報設定時間
(2) ホストプロセッサからの制御命令の実行時間

への影響が考えられる．(1) については，構成情報の設定動作は FlexSword コアが実行中であっても行なうことができ，FlexSword コアが連続して処理を実行する場合，構成情報の設定時間は隠蔽され，FlexSword の性能へは大きく影響しない．また，2 回目以降の構成

情報設定では前回の処理の構成情報との差分のみを指定すればよいため，似た処理を連続して指示する場合は構成情報設定時間を短縮することができる．以上より，(1) の近似の適用による性能への影響は軽微だと考えられる．(2) については，構成情報の設定に複雑な条件分岐が伴う場合などに特に影響は大きくなると考えられる．しかし，本評価ではこの影響について考えない．

処理の重ね合わせについての近似

3.2 節に述べた処理の重ね合わせについて，近似を用いて評価する．近似の内容は以下の通りである．

- 先行処理の実効時間が 32 サイクルより大きい場合，先行処理の終了する 16 サイクル前から後続処理の重ね合わせての動作を開始する．
- 先行処理の実行サイクルが 32 サイクル以下の場合，先行処理の実効時間のうち 50%が経過したら後続処理を開始する．

近似の適用により前後の処理は最大で 16 サイクル重なって動作する．表 1 より，Formatter0 から直接 Write Control へデータを受け渡し，ローカルメモリへデータを書き戻すときのときのレイテンシの合計は 10 サイクルである．そのため，処理を重ねることが出来るサイクル数は少なくとも 10 以上になると考えられる．

本稿では，Formatter0, AUX0，AUX1，Write Control を経由してローカルメモリにデータを書き戻すデータの流れを FlexSword の典型的なデータ処理手順と考える．このような処理を 2 度連続で実行した時に重ね合わせることのできるサイクル数は 16 である．よって，最大で 16 サイクル重なり合うという近似を用いた．

4.2 評価結果

図 8，図 9 に実行結果を示す． 図の縦軸は信号処理性能を，横軸は FlexSword の数を表す．いずれの場合もホストプロセッサ数は 4 である．つまり，FlexSword が 1 つの場合 4 つのホストプロセッサで 1 つの FlexSword を，FlexSword が 2 つの場合 2 つのホストプロセッサで 1 つの FlexSword を共有する．

図より FlexSword 数とホストプロセッサ数の比が 1:1 である場合と比べて，1:4 の場合に信号処理性能が低下していることがわかる．処理の重ねあわせを行わない場合，性能は Cycle で約 9%，Preakness では約 20% の低下が見られる．しかし 1:2 の場合の性能低下はそれぞれ 2%，6%となっており，それほどの悪化は見られない．また，1:4 の共有モデルについても，処理の重ね合わせを用いることでその性能低下を抑えることができる．処理の重ねあわせを行ったとき，1:4 の共有での性能低下はそれぞれ約 5%，約 10%となっている．

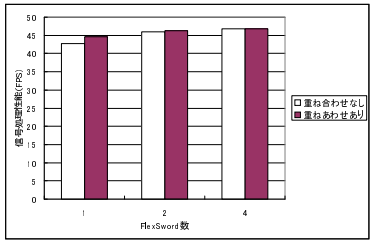

図 8 FlexSword 共有時の信号処理性能 (Cycle)

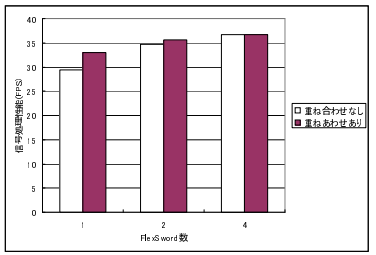

図 9 FlexSword 共有時の信号処理性能 (Preakness)

よって，FlexSword を複数のホストプロセッサで共有しても，それほど大きな性能の低下は引き起こさないと考えられる．さらに，処理の重ねあわせを行うことにより性能の低下を抑えることができるが，処理の重ね合わせの効果については後述する． 図 10，図 11 は同様

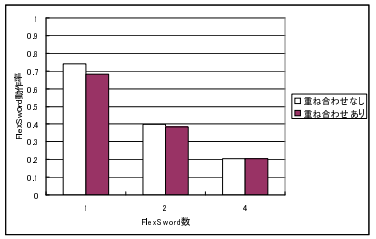

図 10 FlexSword の動作率 (Cycle)

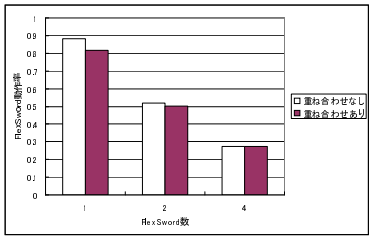

図 11 FlexSword の動作率 (Preakness)

の条件における FlexSword の平均の動作率を示したものである．FlexSword の動作率とは全信号処理時間中の FlexSword 動作時間の割合である．各ホストプロセッサが FlexSword を保持する場合，各 FlexSword は 20%程度しか動作しない．

以上の結果から，FlexSword には共有を行うことのできるだけのアイドル時間が存在し，共有の結果 FlexSword の動作率は向上し，アイドル時間が削減されることがわかる．FlexSword 動作率は図 10，図 11 では最大 88%まで上昇している．しかし，動作率が非常に高い場合，FlexSword での処理が処理全体のボトルネックになることが考えられるので，適切な数のホストプロセッサでの共有を行わなければならない．

次に，1 つの FlexSword を共有するホストプロセッサ数について評価する．図 12，図 13

に 1 つの FlexSword を共有するホストプロセッサの数を変化させたときの性能の変化を示す． ここで非共有とは，FlexSword を共有せず，各ホストプロセッサがそれぞれ専用の

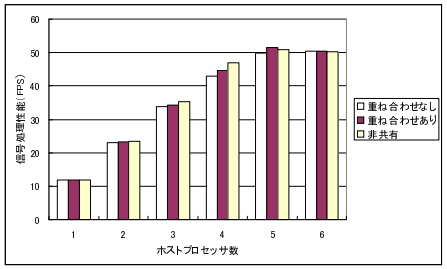

図 12 ホストプロセッサ数と信号処理性能 (Cycle)

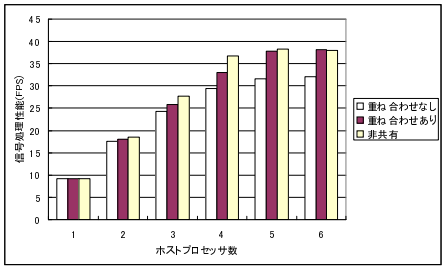

図 13 ホストプロセッサ数と信号処理性能 (Preakness)

FlexSword に接続している場合の性能を表す．

Cycle の場合，ホストプロセッサ数が 5 以下では，ホストプロセッサ数の増加に対して信号処理性能も増加している．非共有の場合と比較して 4 ホストプロセッサのときに最大の約 20%の性能低下があるが，処理の重ね合わせを行うことによって 10%程度まで性能低下を抑えることが出来る．しかし，ホストプロセッサ数が 5 以上ではホストプロセッサ数が増えても性能は向上していない．FlexSword を共有しない場合も性能向上していないので，FlexSword の共有によるものではなく，別の要因によるもの考えられる．Preakness の場合もほぼ同様であるが，重ねあわせを行わない場合ホストプロセッサ数が 4 以上で性能向上が少なくなっている．一方，非共有モデルではホストプロセッサ数が 5 までは性能向上しているため，ここでは FlexSword の共有により性能向上が妨げられていると考えられる．

図 14，図 15 にこの時の FlexSword 動作率を示す．共有するホストプロセッサが多いほど，FlexSword の動作率は向上していることがわかる．重ねあわせを行わない場合，動作率の最大値は Cycle で約 86%，Preakness で約 88%である．図 15 を見ると，重ね合わせを行わない場合，ホストプロセッサ数が 4 以上の場合 FlexSword の動作率はほぼ一定である．図 13 とあわせて考え，FlexSword の動作率が飽和しているために全体の性能も向上しなったと考えられる．

以上より，FlexSword の動作率は 88%程度が上限であり，それ以上の負荷がかかるほどの共有を行った場合，FlexSword が処理のボトルネックとなり得る，ということが言える．

また，ホストプロセッサが 5 以上のときに性能向上を阻害する要因としては，

- スレッド生成，管理のボトルネック化
  スレッドの実行，管理を行う制御プロセッサがボトルネックとなり性能が向上しない
- スレッド分割による並列性抽出の限界
  アプリケーションを複数のスレッドに分割した際に十分な並列性が確保できず，多数のプロセッサで並列実行した際にその資源を有効利用できていないため性能が向上しない

などが考えられるが，正しい原因の解明にはさらなる調査が必要である．

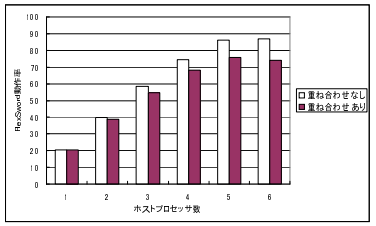

図 14 共有ホストプロセッサ数に対する FlexSword 動作率 (Cycle)

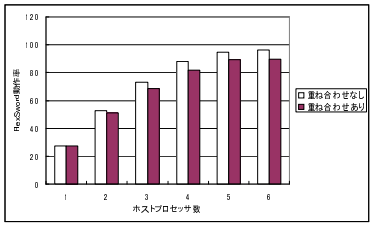

図 15 共有ホストプロセッサ数に対する FlexSword 動作率 (Preakness)

次に，処理の重ね合わせの効果について評価する．図，12，図 13 などを見てもわかるように，重ね合わせの適用により FlexSword 共有による性能の低下を抑えることが出来る．特に多くのホストプロセッサにより共有した場合，その効果も大きくなる．図 14，図 15 では FlexSword 動作率を 10%以上低下させることが出来ている．図 16，図 17 にホストプロセッサが処理開始の指示を出してから実際に FlexSword コアにおいて処理が始まるまでの待機時間を示す．FlexSword を共有するホストプロセッサの数が増えるほど FlexSword の稼働率は上昇し，それに伴い実行開始までの待機時間も増加している． この時，処理の重ね合わせの適用によって FlexSword 待機時間は半分以下になっている．重ね合わせを用いることで多くのホストプロセッサでの共有を行っている場合でもより少ない待機時間で FlexSword 上での処理を開始できることがわかる．

5. 考　　察

4 章では複数のホストプロセッサが FlexSword を共有したときの性能の変化について評価した．本章では共有による面積効率の向上について考える．

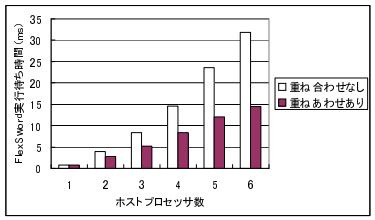

図 16 FlexSword コア実行待ち時間 (Cycle)

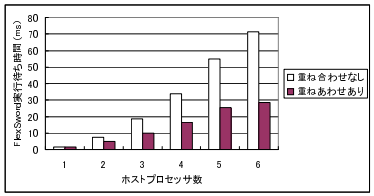

図 17 FlexSword コア実行待ち時間 (Preakness)

5.1 面積効率モデル

複数プロセッサが FlexSword を共有するシステムの面積 S を次式のように表す.

$$S = n_{share}n_{FScore}(S_{host} + S_{if}) + n_{FScore}S_{FScore} + S_{other} \quad (1)$$

ここでそれぞれの記号は

- n_{share}：1 つの FlexSword コアを共有するホストプロセッサ数
- n_{FScore}：FlexSword コア数
- S_{host}：ホストプロセッサの面積
- S_{if}：FlexSword インタフェース，ローカルメモリの面積
- S_{FScore}：FlexSword コア及びアービタの面積
- S_{other}：その他の回路の面積

を表す．また，本稿では面積効率を次のように定義する．

面積効率 = 信号処理性能 (FPS) / 回路規模 (K ゲート)

以下この式に従い面積効率を評価する．

5.2 面積効率評価

面積効率の式を実際の回路に適用し，FlexSword 共有による面積効率の変化について評価する．なおここでは，面積に関してゲート数を用いて評価する．また，ホストプロセッサとして，本稿では ARM 社の ARM9 TDMI[3] を考える．ARM9 TDMI は 5 段パイプラインを備えた 32 ビット RISC プロセッサである．また，4KB ずつの命令キャッシュ，データキャッシュを持つ．L2 キャッシュについては評価に含めない．ARM9TDMI を含め，今回評価に用いる回路の回路規模の概算値をを表 3 に示す.

なお，本稿では処理の重ね合わせによる回路規模の増加についての評価を行っていないため，以後の評価は全て処理の重ね合わせを行わない場合のものを対象とする．

まず，共有を行わない場合の面積効率を図 18，図 19 に示す．ホストプロセッサ数 1，

表 3 各回路の面積 (回路規模)

	回路規模 (K ゲート)
ARM9 TDMI	320
FlexSword コア	658
FlexSword インタフェース	50
FlexSword ローカルメモリ	60

FlexSword 数 1 のときの面積効率を 1 として正規化している． この時，ホストプロセッサ

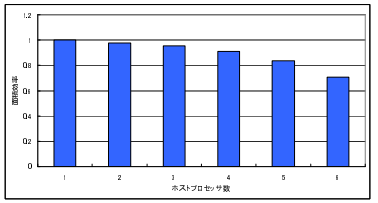

図 18 FlexSword 非共有時の面積効率 (Cycle)

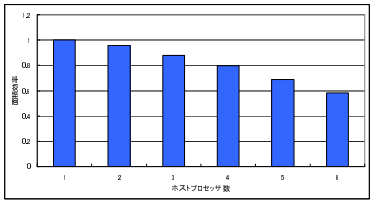

図 19 FlexSword 非共有時の面積効率 (Preakness)

数，FlexSword が増えるほど面積効率は低下している．Preakness では 4 ホストプロセッサの場合約 20%，6 ホストプロセッサでは 42%の面積効率の悪化が見られる．このように FlexSword を共有しない場合，マルチコア環境で各ホストプロセッサが FlexSword を保持することは得られる性能向上に対して大きな面積の増加を招いてしまう.

次に，FlexSword 共有時の面積効率の評価結果を図 20，図 21 に示す．これは FlexSword の数を 1 に固定した状態でホストプロセッサの数を変化させたときの面積効率を比較したものである.

Cycle, Preakness 共に 1 つのホストプロセッサ，1 つの FlexSword の構成と比べ，複数のホストプロセッサで FlexSword を共有した場合面積効率は良くなっている．また，Cycle の場合はホストプロセッサ数 4，Preakness の場合はホストプロセッサ数 3 のときにそれぞれ面積効率の評価値は最大となっている．この時，1 ホストプロセッサ，1FlexSword の時と比べてそれぞれ約 1.7 倍，1.5 倍となっている．ホストプロセッサ数が 5 以上の時，面積効率も低下しているが図 12，図 13 における信号処理性能も同様のホストプロセッサ数であったことから，性能が飽和しているためと考えられる.

逆に，ホストプロセッサ数が 4 以下の範囲では面積効率は共有ホストプロセッサが増える

Vol.2009-ARC-184 No.16
2009/8/5

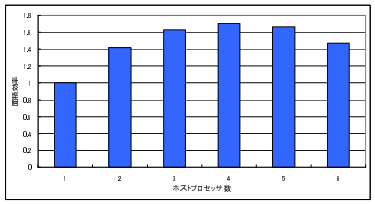

図 20 面積効率評価結果 (Cycle)

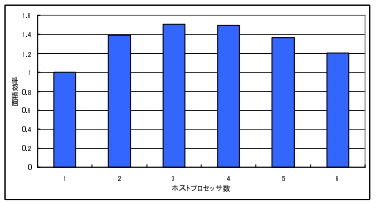

図 21 面積効率評価結果 (Preakness)

ほどよくなっている．つまり，FlexSword 共有により性能が低下する以上に面積を削減できていると考えられる．

6. おわりに

本稿では動的再構成エンジン FlexSword を用いたシステムにおいて，複数のプロセッサで 1 つの FlexSword を共有する手法を提案した．FlexSword の共有により FlexSword の動作率は上昇し，わずかな性能の低下に対して大きな面積削減効果を得られる．FlexSword 非共有時と比較して，1 つの FlexSword を 4 つのホストプロセッサで共有した時，最大で 1.7 倍の面積効率の向上を確認できた．

また，FlexSword の共有時に複数の処理を重ね合わせて実行することにより，ホストプロセッサが指示を出してから FlexSword が処理を開始するまでの待機時間を削減し，FlexSword 共有時の性能低下を抑えることができることを確認した．処理の重ね合わせの実施により FlexSword 処理開始までの待機時間を半分以下に短縮できた．

今後の課題としては，処理の重ね合わせのハードウェア上での実装及び面積，性能への影響の評価などが挙げられる．また，本稿での評価は単一のアプリケーションのみで行ったため，他のアプリケーションを実行した場合の面積効率，および最適な FlexSword 共有ホストプロセッサ数の評価についても考えなければならない．

参 考 文 献

1) Takashi Yoshikawa, Yutaka Yamada, Shigehiro Asano, “An Implementation of hardware accelerator using dynamiccaly reconfigurable architecture”, IEEE HotChips, Aug 2006.
2) Yutaka Yamada, Takashi Yoshikawa, Shigehiro Asano, “Implement and Evaluation of the processor for stream Multimedia Applications using Dynamic Reconfiguration”, IEEE COOL Chips X, April 2007.
3) Simon Segars, “The ARM9 Family - High Performance Microprocessors for embedded Applications”, International Conference on Computer Design (ICCD)’98, pp.230-235.