# 組込みタイルプロセッサ-TEMP-の設計とマルチメディア演算の実装

田路　真也† 森　秀樹† 上原　稔†
東洋大学大学院　工学研究科　情報システム専攻†

## 1．はじめに

昨今、急速なマルチメディアコンテンツの増加に伴い高度なマルチメディア処理が求められている。組み込みの世界においてもこの流れは顕著で DVD プレイヤーや携帯音楽プレイヤー、またテレビゲーム機や携帯電話などは高速な処理が必要となってきている。そこで本稿では汎用のプロセッサに追加するマルチメディア用サブプロセッサ-TEMP-を提案する。本稿においてはタイルプロセッサにバタフライ演算を実装させ命令速度の高速化を図った。

## 2．プロセッサ仕様の決定

プロセッサのアーキテクチャとしてタイルアーキテクチャを採用した。タイルアーキテクチャとは演算タイルを格子状に並べ隣接したタイルのみと通信を行なうことにより配線遅延を回避するアーキテクチャである[1]。また配線遅延回避のほかに高い並列性も持っているのが特徴である。このタイルプロセッサの中でも今回はタイルプロセッサにデータフローアーキテクチャの要素を足したテキサス大学で研究中の TRIPS[2]と呼ばれるタイルプロセッサを参考にすることにした。

データフローアーキテクチャとは非ノイマン型のアーキテクチャでその計算方法はグラフで表現される。グラフのノードに配置された各演算に対してトークンとして演算に必要なデータを受け取り次第演算を実行(発火)していく仕組みである[2]。

TRIPS のコンパイラは各タイルに静的に命令を配置する．各タイルはデータが到着し準備ができ次第命令を実行していく。 実際には動的に動作はしていないが、データと命令がそろって初めて演算が行われる TRIPS の仕組みはデータフローアーキテクチャの考え方に基づいたものである。TRIPS では命令やデータをタイルの周囲のキャッシュに配置しているため、命令やデータをフェッチするために遅延が発生する。この遅延を隠蔽するために一度に複数の命令をフェッチしている。

今回作成するプロセッサ TEMP は 8bitALU2 個と 8bit データレジスタが 4 個、16bit 命令レジスタ、ルータからなる演算タイルを図 1 のように 20 個、5×4 の格子状に並べた。ただし各タイルが接続しているタイルは TRIPS が上下左右なのに対して TEMP は左右と下、さらに斜め右下と斜め左下のタイルとも接続している。

メインプロセッサとは共有のデータレジスタを介してデータのやりとりをする。そしてタイルプロセッサはメインプロセッサからもらったデータと命令によって各タイルに命令、また必要なデータを格納していく。あとは演算タイルが計算に必要なデータが到着しだい計算をし、マルチメディア処理を実行する。

"A Design of a Tile-based Embedded Multimedia Processor - TEMP- and implementation of a multimedia operation"
Shinya Toji †, Hideki Mori†, Minoru Uehara†
Graduate School of Engineering, Toyo University
Dept. of Open Information Systems

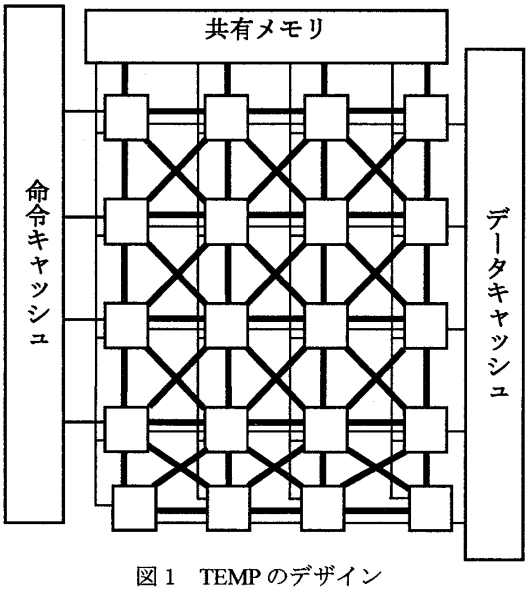


図 1　TEMP のデザイン

## 3．命令の決定

はじめにタイルプロセッサにどのようなマルチメディア命令を実装するかを考えた。高速化の手段としてはまず典型的なマルチメディア拡張命令に使われる SIMD 命令を実装することを考えた。SIMD(Single Instruction Multiple Data)命令とは命令はひとつの命令で複数のデータの処理を行うことのできる命令のことである。しかし通常の SIMD 演算だとタイルに無駄ができてしまう。そこで今回は SIMD 命令よりさらに複雑なバタフライ演算を実装することにした。

### 3－1 バタフライ演算

バタフライ演算とは FFT で行なう演算で $N$ をサンプル数、$W_N^k = \exp(-j2\pi k/N)$ とすると以下の式で表せる。

$$X(k) = x(k) + W_N^k \times x(k+\frac{N}{2})$$

$$X(k+\frac{N}{2}) = x(k) - W_N^k \times x(k+\frac{N}{2})$$

バタフライ演算は入力がたすき掛けになっており図 2 に示すように図で表すと蝶の羽のような形になることからバタフライ演算と名付けられた。

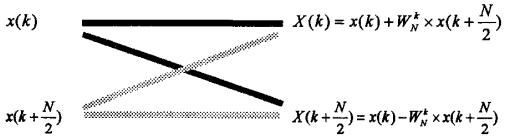


図 2　バタフライ演算

$N=8$ の場合このバタフライ演算を 12 回行なうことによって FFT の結果を得ることができる。
1 個のバタフライ演算は 1 回の複素数乗算と 2 回の複素数加減算からなっており計算を $N/2\log_2 N$ 減らすことができる。

### 3-2 演算タイルにおける命令の決定

バタフライ演算を実装するために演算タイル自体の命令を以下のように決定した。
■タイルの実行する命令は基本的な命令に加え即値命令と複素数乗算用命令を追加。
■命令長はオペコード 4bit と出力先オペランド 4bit、さらに即値用に 8bit 用意する。

## 4. タイルプロセッサのシミュレーションとマルチメディア命令の実装

前章で決めた仕様に基づいて演算タイル tile を Verilog にて設計し論理合成、シミュレーションを行なった。

そしてこの演算タイルを 20 個格子状に並べたタイルプロセッサを設計し、マルチメディア命令を実装させた。

今回実装したバタフライ演算命令は 2 個のバタフライ演算を平行に行い、一回の割り当てで計 4 個のバタフライ演算を行なうことができる。各タイルへの割り当て方を図 3 に示す。

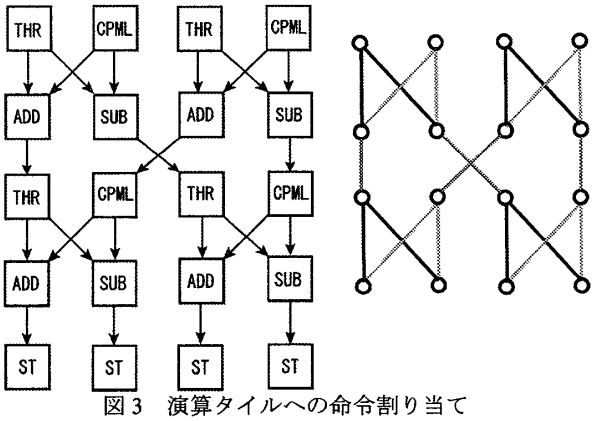

図 3　演算タイルへの命令割り当て

次にこのバタフライ演算を使い 8 点 FFT の割り当てを考えた。今回の割り当てでは 4 つのデータを使いバタフライ演算を 4 回実行している。そこでまずはこの割り当てを 2 回行うことによって 8 つのデータに対して 2 回ずつバタフライ演算を行ったことになる。

残りは各 1 回ずつ行うと 8 点 FFT になるがこの場合結果のデータの位置がバラバラになっているためにこのままでは今までと同じようにバタフライ演算を行うことができない。そこでデータの位置を変えるためデータをクロスさせる割り当てをつくりデータの配置を変更することによりこの問題を解決した。今回の FFT の場合は 1 回の割り当てでデータ移動を行うことができる。こうしてデータを移動した後残りのバタフライ演算を行うことによって 8 点 FFT を行うことができる。

今回のタイルプロセッサにおける 8 点 FFT 計算の流れを図 4 に示す。

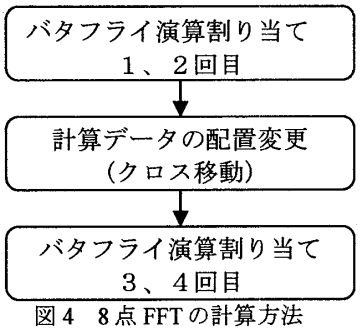

図 4　8 点 FFT の計算方法

また今回のタイルプロセッサと一般的な命令を持つ汎用のプロセッサの 8 点 FFT のステップ数を表 1 に示す。

表 1　バタフライ演算におけるステップ数の比較

| | タイルプロセッサ | 汎用プロセッサ |
|---|---|---|
| バタフライ演算 | 18 ステップ | 144 ステップ |

この表にあるように急激なステップ数の減少がみられた。これはバタフライ演算を並列に行なっているのと複雑な複素数乗算をひとつの命令にまとめて処理していること、実部と虚部を同時に計算していることなどが要因として挙げられる。

しかし今回の 8 点 FFT ではデータ移動が一回の割り当てですんだがサンプル数が多くなるにつれデータ移動の回数も多く複雑になるために多くの割り当てが必要になると予想される。また今回の作成したタイルプロセッサではタイルプロセッサにおいて一番時間のかかる命令フェッチを省略している。そこで今後は命令フェッチの遅延を考えたより実物に近いプロセッサを作っていく必要がある。

## 5. おわりに

本稿ではマルチメディア処理用のサブプロセッサとしてタイルプロセッサにバタフライ演算を実装させた。プロセッサのデザイン、命令を決定し、効率的なタイルへの割り当て方を提案した。同時に複数のバタフライ演算を行なうことにより結果としてステップ数の大幅な減少が見られた。今後は今回作成したタイルプロセッサを使った具体的な FFT の計算方法、またそれに伴うメモリアドレッシング方法などを決めていくことが課題として挙げられる。そして最終的にタイルプロセッサを組み込む前後での演算処理速度の向上率、既存のマルチメディアプロセッサとの処理速度の比較などを行なう予定である。

参考文献

[1] 吉瀬謙二 「新世代マイクロプロセッサアーキテクチャ（前編）:1.アーキテクチャ基盤技術 6.タイルプロセッサ」情報処理学会誌「情報処理」 2005 年 10 月号
[2]J.A.シャープ著／富田眞治訳 「データ・フロー・コンピューティング」 サイエンス社 1987 年
[3] "Exploiting ILP, TLP, and DLP Using Polymorphism in the TRIPS Architecture," K. Sankaralingam, R. Nagarajan, H. Liu, J. Huh, C.K. Kim D. Burger, S.W. Keckler, and C.R. Moore. 30th Annual International Symposium on Computer Architecture (ISCA), pp. 422-433, June 2003.