2003－ARC－153 (9)
2003／5／8

マルチスレッド技術を用いたマルチメディア処理向けベクトルユニットの設計と実装

松浦克彦[†] 伊藤務[†] 山崎信行[†]

本論文では，*Responsive Multithreaded*（*RMT*）*Processor* 上に設計・実装したマルチメディア処理向けベクトルユニットについて述べる．マルチメディア処理では，大量の演算を高速に行うことが要求される．そこで，本研究では 2 次元のデータ並列性を利用した高速な演算を行うことができるベクトルユニットを設計・実装した．また，マルチスレッド技術を用いることでベクトル演算の遅延を隠蔽し，全体の性能を向上させた．さらに，演算ユニットを連結するパイプラインチェイニングを実装し，マルチメディア処理で頻繁に行われる複合演算を高速に行えるようにした．

Vector Unit using Multithreading for Multimedia

Katsuhiko Matsuura,[†] Tsutomu Itou[†]
and Nobuyuki Yamasaki[†]

This paper describes the vector unit for multimedia processing designed and implemented on the *Responsive Multithreaded (RMT) Processor*. In multimedia processing, it is required that a large amount of operation should be performed at high speed. Then, the vector unit which can perform high-speed operation using 2-dimensional data parallelism. Moreover, delay of vector operations were concealed by using multithreading technology, and the whole performance was raised. Furthermore, pipeline-chaining which connects execution units is implemented and it enabled it to perform compound operation frequently performed in multimedia processing at high speed.

1. はじめに

マルチメディアアプリケーションにおける画像処理や音声処理にはリアルタイム性が要求される．制御アプリケーション等と比較すると時間制約は厳しくないが，画像処理や音声処理では，大量のデータに対して高速に演算を行わなければならない．それゆえ，マルチメディア処理の高速化はソフトウェア，ハードウェアの両面からさまざまに行なわれている．ソフトウェアの面からは，演算を減らし，処理を高速に行うためのアルゴリズムの研究が行われているが，基本的なアルゴリズムを変更することは難しい．一方ハードウェアでは，マルチメディア処理で行われる演算の特徴を考慮した高速化の手法により著しい成果をあげている．本研究では，ハードウェアによりマルチメディア処理を高速化することを目的とする．

大量のデータを高速に処理するためには，SIMD（Single Instruction Stream Multiple Data Stream）命令やベクトル演算の使用が有効である．これらのアーキテクチャではデータの並列性を活用し，複数のデータに対して同じ演算を高速に行うことができる．また，MOM（Matrix Oriented Multimedia Extension）[1] は，SIMD 命令とベクトル演算を融合することにより，データ並列性を 2 次元に拡張している．画像処理等のマルチメディア処理では行列演算が本質的であるため，2 次元のデータ並列性を活用した演算をハードウェアで実現することは有効であると考えられる．また，マルチメディアアプリケーションでは画像と音声のように複数のデータストリームを同時に処理することが必要となる．

そこで本研究では，複数のスレッドを同時に実行できるマルチスレッドプロセッサ上に MOM を応用したベクトルユニットを設計・実装し，マルチメディア処理を高速化する．

† 慶應義塾大学
Keio University

2. 背　　景

2.1 マルチメディア処理

マルチメディア処理では多くの場合，ビット幅の小さい大量のデータ（画像のピクセルに 8 ビット，音声のサンプリングデータに 16 ビット等）に対して，同じ演算を繰り返す．また，扱うデータには十分な並列性がある．例えば，MPEG 符合化では 8×8 のブロック（各ピクセルは 8 ビットで表される）に対して，離散コサイン変換を繰り返し行う．このとき，各ブロックを並列に処理できる．また，画像についてはピクセル単位で扱われるが，通常のデータと違い，ピクセル間に 2 次元的な相関関係があるため，行列演算が本質的である．さらに，マルチメディア処理では積和演算等の複合演算を頻繁に行う．

2.2 マルチメディア処理向けアーキテクチャ

マルチメディア処理を高速化するプロセッサアーキテクチャには，データ並列性を活用した SIMD 命令とベクトル演算がある．また，活用できるデータ並列性を 2 次元に拡張し，行列演算を可能にした MOM がある．本節では，これらのアーキテクチャについて述べる．

SIMD 命令は 1 つの命令で同時に複数のデータに対して同じ処理を行うことにより，マルチメディア処理を高速に行う．マルチメディア処理において，8 ビットまたは 16 ビットの狭いビット幅のデータに対して処理を行う場合，従来の 32 ビット幅のレジスタでは下位の 8 ビットまたは 16 ビットはデータが格納されるが，上位の 24 ビットまたは 16 ビットは無駄になってしまう．そこで，MMX（MultiMedia eXtension）[2] 等の整数 SIMD 命令では，1 つのレジスタに複数の 8 ビットまたは 16 ビットのデータを詰め込み，1 つの命令で一度に処理する．整数 SIMD 命令には，MMX 以外にも MDMX（Mips Digital Media Extension）[3]，VIS（Visual Instruction Set）[4] 等がある．

現在の画像処理では，主に固定小数点式の整数演算が使用されている．しかし，多彩なグラフィックス機能を駆使し高精細画質が要求されるケースが増えてきているため，最近の商用プロセッサには浮動小数点 SIMD 命令も実装されている．SSE（Streaming SIMD Extension）[5] や Altivec[6] 等の浮動小数点 SIMD 命令では，従来の浮動小数点レジスタよりもビット幅の広いレジスタを SIMD 命令専用に用意し，そのレジスタに複数の単精度浮動小数点データを詰め込み，一度に処理する．

ベクトル演算はスーパーコンピュータによって一般化したアーキテクチャである．1 つの命令で複数のデータを処理するという点では SIMD 命令と同じであるが，違いは，SIMD 命令では全ての要素を同時に操作するのに対し，ベクトル演算は通常 1 クロックサイクル当たり数個のベクトル要素を操作するパイプライン化された機能ユニットをベースにしている点である．SIMD 命令ではデータの並列性はレジスタの幅によって制限されるが，ベクトル演算ではベクトル長を長くすれば，それだけ並列性を活用することができる．ただし，ベクトル長に比例して演算にかかる遅延も長くなる．

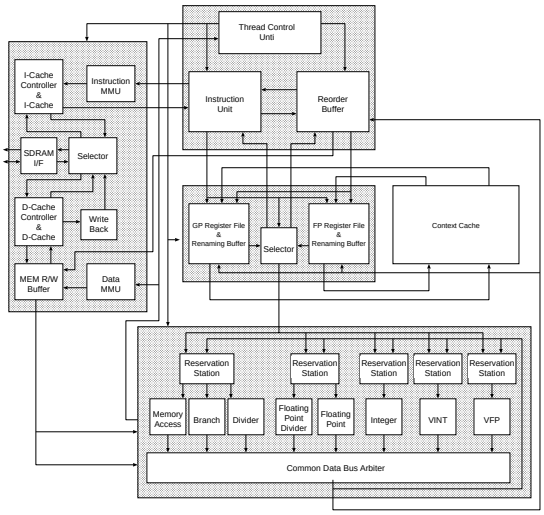

図 1　RMT プロセッサのプロセッシングコア（RMT PU）

従来の SIMD 命令ではレジスタのビット幅にデータの並列性が制限されてしまう．一方，ベクトル演算では 8 ビットや 16 ビットのデータを処理する場合に，上位の使わないビットが無駄になってしまう．

MOM は，従来の SIMD 命令とベクトル演算を融合することで行列構造を高速に処理するアーキテクチャである．MOM では 2 次元のデータ並列性を活用した行列演算を行うことができる．MOM は MMX と同様に複数の 8 ビット，16 ビット，32 ビットのデータを 64 ビットレジスタに詰め込み，さらにそのレジスタ 16 個を 1 つのベクトルレジスタとし，1 つの命令で処理するアーキテクチャである．

2.3 Responsive MultiThreaded Processor

Responsive MultiThreaded（*RMT*）*Processor*[7] は，分散リアルタイムシステムを実現するために，リアルタイム通信およびリアルタイム処理を同時にハードウェアレベルで支援することを目的として設計されたシステムオンチップである．RMT プロセッサの RMT PU（**図 1**）は，優先度付細粒度マルチスレッディング

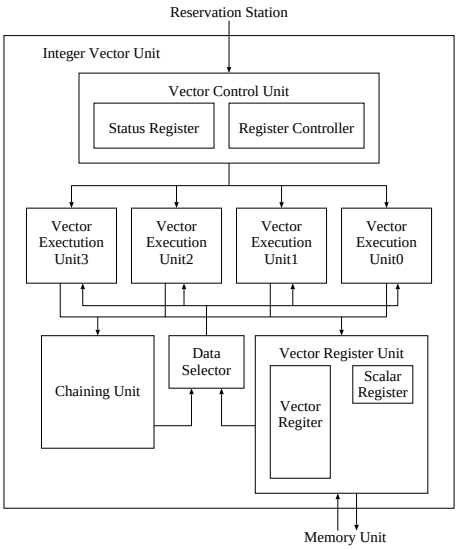

図 2 Vector Unit

技術を用いており，以下のような特徴を有している．

- 8 スレッド同時実行
- ハードウェアによるコンテキストスイッチ
- 256 レベルの優先度を用いた機能ユニットの制御
- 割込み発生時のハードウェアによるスレッドの制御

RMT PU は，これらの機能により細粒度のリアルタイム処理を支援することができる．

3. 設計及び実装

3.1 設計方針

本研究では，MOM の技術を用いた 2 次元のデータ並列性を活用した行列演算を行うことができる整数ベクトルユニットを設計する．また，ベクトルユニットの演算ユニットを複数連結することで，マルチメディア処理で頻繁に行われる積和演算等の複合演算を高速に行えるようにする．さらに，マルチスレッド技術を用いることでベクトル演算の遅延を隠蔽し，システム全体として高い性能を実現する．

3.2 Vector Unit

Vector Unit **図 2** は，1 つの命令でベクトルレジスタの各エントリに対して，同じ演算を繰り返し行う機能ユニットである．この機能ユニットを利用したベクトル演算では，まずベクトル演算を行うスレッドが必要な分だけベクトルレジスタを確保する．それから，そのベクトルレジスタを利用したベクトル演算を行う．ベクトル演算が終了し，ベクトルレジスタが必要なくなると確保していたレジスタを解放する．

Vector Unit は，Vector Register Unit，Vector Control Unit，4 つの Execution Unit，Chaining Unit, Data Selector から構成される．

表 1 確保できるレジスタ領域

ベクトルレジスタ（ベクトル長 × 本数）	スカラーレジスタ（エントリ数）
8 × 16	8
16 × 8	8
8 × 32	16
16 × 16	16
32 × 8	16
16 × 32	32
32 × 16	32
64 × 8	32

以下では Vector Unit の各構成モジュールについて詳しく述べる．

3.2.1 Vector Register Unit

ベクトルレジスタとは，ベクトル長分のエントリを持つレジスタである．上述した MOM では，ビット幅 64，ベクトル長 16 のレジスタを 16 個持っている．マルチスレッドプロセッサにおいて，各スレッドのコンテキストを格納するために，全てのスレッドにこの大きさのレジスタを用意すると非常に大きなものになってしまう（8 スレッド同時実行を可能にするためには，16K バイトのレジスタが必要）．そこで，64 ビット幅の整数レジスタ 512 個（合計 4K バイト）をベクトル演算用のレジスタとして用意し，これを複数のスレッドで共有できるようにする．各スレッドはベクトル演算を行う前に VRES(Vector Reserve) 命令によって必要なだけレジスタを確保する．そのスレッドのベクトル演算が終了し，レジスタが必要なくなったときに VREL(Vector Release) 命令によって確保していたレジスタを解放する．また，ベクトル演算ではスカラーとベクトル間での演算も必要になるため，64 ビット幅のスカラー用レジスタも 32 個用意する．VRES 命令により，ベクトル演算用レジスタとともにスカラー演算用レジスタも確保される．VRES 命令によって確保できるレジスタの構成は，**表 1** の 8 種類である．ベクトル長分のエントリ数を持つレジスタが 1 つのベクトルレジスタである．

VRES 命令により確保できる領域を 128，256，512 エントリに固定することで，レジスタの確保と解放に必要になるハードウェアの複雑化を避けている．

3.3 Vector Control Unit

Vector Control Unit は，Vector Register Controller と Status Register から構成される．Vector Register Controller は，ベクトル演算に必要な各スレッドのベクトルレジスタ情報（エントリ数，全ベクトルレジスタ内での位置等）を管理し，スレッド毎に

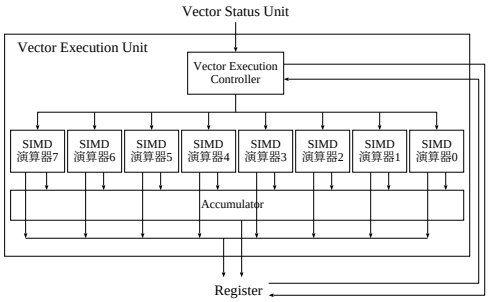

図 3　Vector Execution Unit

異なるベクトルレジスタ ID を生成する．Status Register は各スレッドのベクトル長とマスクビットを管理する．Vector Control Unit はこれらの情報とオペレーションを空いている Vector Execution Unit に発行する．

3.4 Vector Execution Unit

複数のスレッドが並列にベクトル演算を実行できるようにし，高い演算能力を実現するために，Vector Unit には 4 つの演算ユニット（Vector Execution Unit）を用意した．また，複数の演算ユニットを連結させること（パイプラインチェイニング）で，直接別の演算ユニットに演算結果を渡すことを可能にする．

Vector Execution Unit（**図 3**）は Execution Controller，8 つの SIMD 演算器，Accumulator から構成される．

Execution Controller は，Vector Control Unit から発行されるレジスタ ID をもとに Register Unit にアクセスし，データとオペレーションを演算器に供給する．Vector Execution Unit は，8 つの SIMD 演算器を持ち，MPEG 符号化等のマルチメディア処理で頻繁に行われる 8×8 のブロックを基本とした演算に適した構造になっている．この機能ユニットでは，ベクトルレジスタの 8 つのエントリを一度に処理できる．ベクトル長が 16，32，64 の場合は，それぞれ 2，3，4 回同じ操作を繰り返す．このとき，最後の演算が終了するまで演算の対象となるベクトルレジスタを保持しておくと，後続の命令は前の命令が完全に終了するまでストールすることになり，効率が悪い．マルチメディア処理では積和演算等の複合演算を頻繁に行うため，演算結果を後続の命令で使用する場合が多い．そこで，演算ユニットを連結するパイプラインチェイニングを実装し，前の命令の演算結果をレジスタに書き込むと同時に次の命令のソースとして利用できるようにして，複合演算時のストールを低減した．また，各 Vector Execution Unit は Accumulator を持っており，ベクトル長分の演算結果を足し込むことも可能であり，積和演算等を高速に行うことができる．

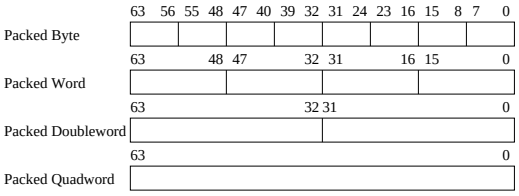

図 4　データタイプ

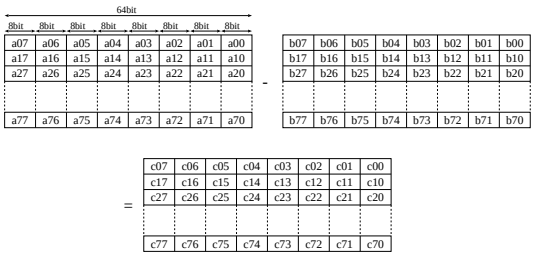

図 5　行列演算の例

各演算器に SIMD 演算器を用いることにより行列演算を可能にしている．各演算器は MMX のように 1 つのレジスタに複数の 8 ビット，16 ビットあるいは 32 ビットのデータをパッキングし，それぞれのデータに対して同じ処理を同時に行うことができる．実装した演算器により処理することができるデータタイプを**図 4** に示す．

これらのデータタイプの扱いを容易にするため，データの詰め込みと並べ替えをプログラマの要求に応じて柔軟に行えるように，複数のパッキング命令を用意した．また，各要素の演算結果がオーバーフローやアンダーフローを起こした場合の最大値，最小値への丸め処理の機能も SIMD 演算器に実装した．演算結果の精度を維持したい場合はプログラマが任意にパッキング命令を利用してデータプロモーションを実行すればよい．

Vector Unit を用いた行列演算の例を**図 5** に示す．8×8 のピクセルブロックの差分を求めるサンプルコードを考えると，図 5 に示すような演算を 1 回行えばよい．本機構を用いた場合，1 つの行列演算命令でこの演算を実行できる．

```
/* サンプルコード */
for( i=0; i<8; i++){
    for( j=0; j<8; j++ ){
        c[i][j]=a[i][j]-b[i][j];
    }
}
```

通常の整数演算ユニットでは 1 つの命令では 1 組のデータしか処理できないため，上記の演算を行うためには 32 命令必要である．また，MMX やベクトル演算を利用した場合でも 8 命令は必要である．

4. 評価および考察

本機構の評価のために，本機構を用いて 8×8 ピクセルブロックに対して離散コサイン変換を行うプログラムを作成した．このとき，ピクセルデータは 8 ビット，用いるベクトルレジスタのタイプは 8×16，データタイプは Packed Word とした．評価には RTL シミュレーションを用いて，実行時間（クロックサイクル数）を測定した．

4.1 複数スレッド実行時の評価

RMT PU は 8 つのスレッドを同時に実行することが可能である．また，離散コサイン変換は，各ピクセルブロックに対して並列に実行することができる．そこで，複数のスレッドで離散コサイン変換を分担して実行し，実行時間を測定した．ただし，各スレッドの優先度は全て同じとした．比較のために，ベクトル演算を用いない場合の実行時間も測定した．

図 6 および**図 7** の結果より，Vector Unit により大幅に処理能力を向上できたことがわかる．Vector Unit では，データの並列性を十分に活用した演算を行うことができ，ピクセルブロックの処理に非常に適している．さらに，Vector Unit は Accumulator を持っており，ベクトルレジスタのエントリを足し込むことができる．これは，離散コサイン変換で行われる積和演算を効率よく実行することを可能にし，大幅な性能向上につながったと考えられる．

SIMD 演算器を用いない従来のベクトル演算器との比較も行った結果，約 2 倍の性能向上が得られた．これは，Vector Unit では 1 つのベクトルレジスタのエントリに 2 個のデータを格納して演算を行ったためである．精度があまり要求されない処理等では，4 個あるいは 8 個のデータを格納して処理すれば，さらに性能向上が得られる．

図 6 および図 7 の評価結果からは，ベクトル演算を用いた場合とベクトル演算を用いない場合の両方とも，複数スレッドで同時に実行することで，実行速度が向上していることがわかる．これは，各ピクセルブロックのデータ並列性とマルチスレッド技術による命令レベルの並列性の両方から得られたものである．ただし，ベクトル演算を用いて 1 スレッドで実行した方が，ベクトル演算を用いないモデルの 8 スレッドで実行した場合よりも実行時間は短くなった．これは，ブロックレベルのデータ並列性およびマルチスレッド技術により得られる命令の並列性よりも，ベクトル演算によって得られるピクセルレベルのデータ並列性の方が大きいためである．

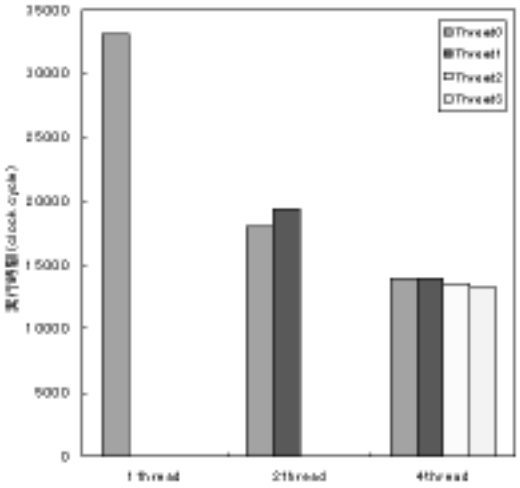

図 6 Vector Unit を用いた場合の離散コサイン変換の実行時間

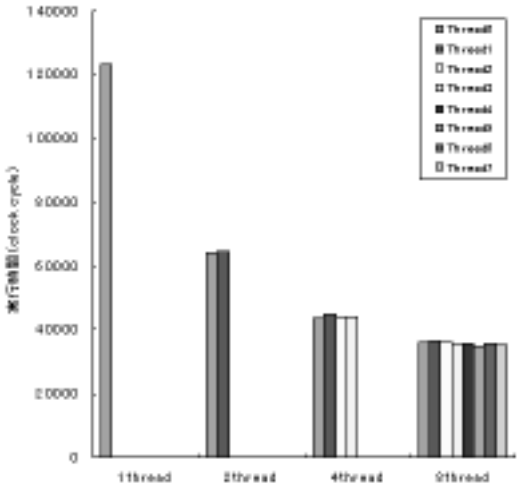

図 7 Vector Unit を用いない場合の離散コサイン変換の実行時間

4.2 優先度を用いて実行した場合の評価

RMT PU では，優先度を用いて機能ユニットの制御を行う．そこで，Vector Unit を利用して離散コサイン変換を行う 1 つのスレッドと優先度の異なるスカラー演算（実際の処理内容は同じ要素数の単純選択ソート）を行う 7 つのスレッドを同時に実行した場合の評価を行った（**図 8** スレッド番号が小さい方が高優先度）．離散コサイン変換を行うスレッドの優先度を

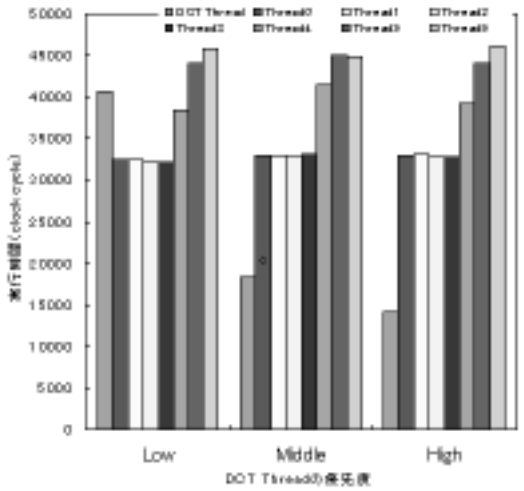

図 8　優先度を用いた場合の実行時間

最低優先度，8 スレッド中で 4 番目の高優先度，最高優先度と変えて実行時間を測定した（図 8）．離散コサインを行うスレッド（DCT Thread）は高優先度時には非常に短い時間で実行を終えている．このとき，DCT Thread が優先して実行されているにも関わらず，他のスレッドはそれほど影響を受けていない．これは，DCT Thread が Vector Unit を主に利用し，少ない命令数で処理を実現するため，他のスレッドと競合する場面が少ないからである．通常，マルチメディア処理は制御等のアプリケーションよりも優先度は低い場合が多く，他の高優先度のスレッドに影響を与えず，高速に実行できることは大きな利点であると言える．

5. 結　　論

本研究では，行列演算により大量のデータを高速に処理するベクトルユニットを RMT プロセッサ上に設計，実装した．これにより，2 次元のデータ並列性を利用した演算を可能にし，ベクトル演算よりも高い演算能力を実現した．また，パイプラインチェイニングを実装したことにより，マルチメディア処理で頻繁に行われる積和演算などの複合演算時のストールを低減できた．さらに複数スレッドでベクトルレジスタを共有することにより，従来のベクトルユニットよりも小さい面積で複数スレッドによるベクトル演算を実現した．

参 考 文 献

1) Corbal, J., Espasa, R. and Valero, M.: Exploiting a New Level of DLP in Multimedia Applications, *International Symposium on Microarchitecture* (1999).
2) Peleg, A., Wilkie, S. and Weiser, U.: Intel MMX for multimedia PCs, *Communications of the ACM*, Vol. 40, No. 1, pp. 24–38 (1997).
3) Technologies, M.: *MIPS Extension for Digital Media with 3D*, MIPS Technologies, Inc., http://www.mips.com/arch/ISA5/MDMXindx (1997).
4) Tremblay, M., O'Conner, J., Narayanan, V. and He, L.: VIS Speeds New Media Processing, *IEEE Micro*, Vol. 16, No. 4, pp. 10–20 (1996).
5) Raman, S. K., Pentkovski, V. and Keshave, J.: Implementing Streaming SIMD Extensions on the PentiumIII Processor, *IEEE Micro*, Vol. 20, No. 4, pp. 47–57 (2000).
6) Diefendorff, K., Dubey, P. K., Hochsprung, R. and Scales, H.: Altivec Extension to PowerPC Accelerates Media Processing, *IEEE Micro*, Vol. 20, No. 2, pp. 85–95 (2000).
7) 山崎信行, 堀俊夫: 分散リアルタイムネットワーク用プロセッサとその応用, 情報処理, Vol. 44, No. 1, pp. 6–13 (2003).